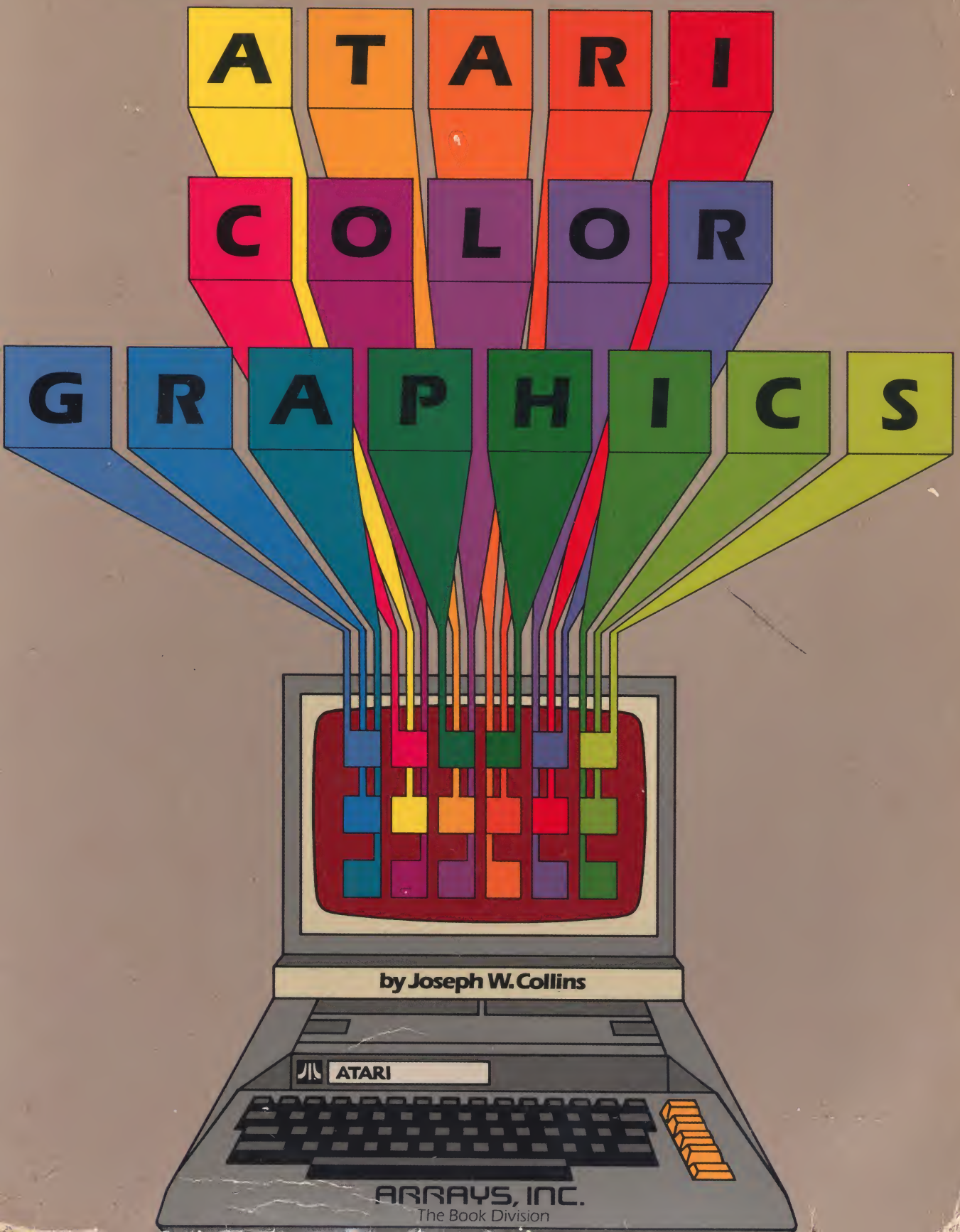
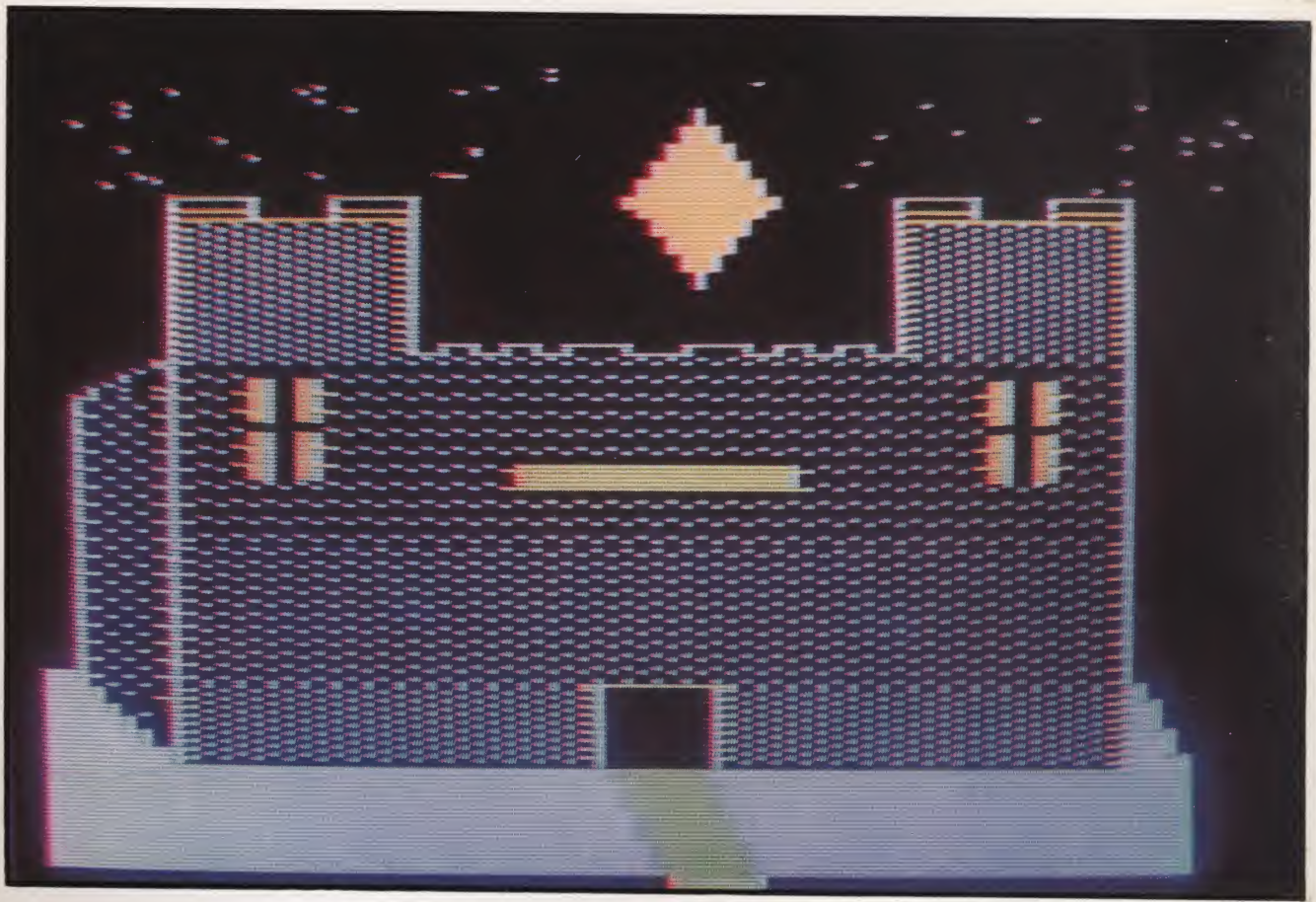


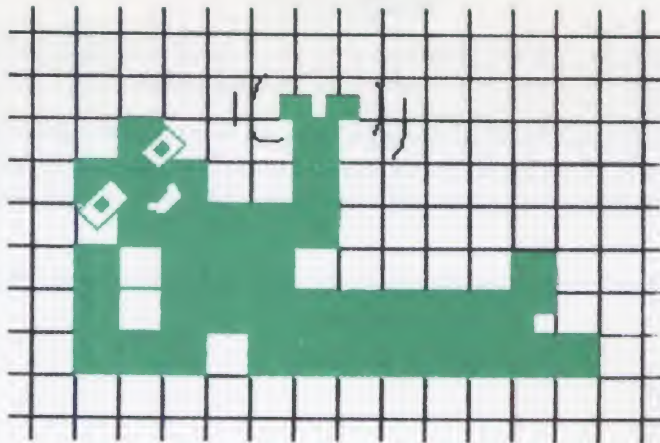
A BEGINNER'S WORKBOOK





frontispiece

ATARI COLOR GRAPHICS: A BEGINNER'S WORKBOOK



By
Joseph W. Collins

ARRAYS, INC.

ARRAYS, INC./THE BOOK DIVISION
11223 So. Hindry Ave. • Los Angeles, CA 90045

Dedication

This book is dedicated to Christine, Vincent, Laurie, and Darren Collins, four nice kids who put up with a lot while their dad was preparing this book.

Also, I dedicate this book to my wife, Kathy, who understands that I am an early riser (4 to 5 A.M.) with tunnel vision. She encouraged me to write the book and pointed out changes she felt were necessary. She's my super-helper and my super-wife.

Finally, the book is dedicated to you, the reader. Without you, there would be no book.

Executive Editor

Michael Mellin

Editor-in-Chief

Roberta M. Ritz

Production Editor

Robert A. Sandberg

Proofreader/Editor

Mark V. Bodziak

Production

Steve Gunn

Mark E. Mansell

Estela Montesinos

Argelia Navarrete

Photography

Richard West

Technical Assistance

Peter A. Densmore

Steve Golding

ISBN 0-912003-19-7

Copyright ©1984 Arrays, Inc./The Book Division. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Arrays, Inc.

Atari is a registered trademark of Atari, Inc. The use of trademarks or other designations is for reference purposes only.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	Page 4
INTRODUCTION	Page 5
HINTS	Page 5
Chapter One	
EDITING	Page 7
Chapter Two	
GRAPHICS MODES	Page 19
Chapter Three	
DISPLAYING CHARACTERS	Page 27
Chapter Four	
GRAPHICS ZERO	Page 33
Chapter Five	
THE TEXT WINDOW	Page 49
Chapter Six	
GRAPHICS ONE AND TWO	Page 57
Chapter Seven	
GRAPHICS THREE, FIVE, AND SEVEN	Page 79
Chapter Eight	
GRAPHICS FOUR AND SIX	Page 101
Chapter Nine	
GRAPHICS EIGHT	Page 107
NOTE TO NON-GTIA AND NON-XL OWNERS	Page 118
NOTE TO GTIA AND XL OWNERS	Page 119
Chapter Ten	
GRAPHICS NINE	Page 121
Chapter Eleven	
GRAPHICS TEN	Page 133
Chapter Twelve	
GRAPHICS ELEVEN	Page 145
Chapter Thirteen	
GRAPHICS FOURTEEN	Page 159
Chapter Fourteen	
GRAPHICS FIFTEEN	Page 169
Chapter Fifteen	
VARIABLES	Page 175
Chapter Sixteen	
MULTIPLICATION SLAVE	Page 181
AFTERWORD	Page 193
GLOSSARY	Page 195

Acknowledgements

The book that came with my first Atari 800, *Atari BASIC—A Self-Teaching Guide*, is an excellent introduction to the computer. Many things I learned from this book are not covered in any other reference source. I was disappointed at first because it didn't teach me everything that I wanted to know; since then, I've read a number of books and articles and I recognize the monumental task the authors undertook. Albrecht, Finkel, and Brown (authors of the above-mentioned work) introduced me to my machine, but it was up to me to read other articles and books and gain more expertise at using my microcomputer.

Another superior introduction to learning BASIC is *Atari Games and Recreations* by Kohl, Kahn, Lindsay, and Cleland. I bounced back and forth between this work and the *Self-Teaching Guide* for months.

A third fine book is *Your Atari Computer* by Poole, McNiff, and Cook from which I learned much useful information. The magazines *COMPUTE!*, *Analog*, and *Antic* have all been useful as well.

The Book Company publishes three superb books about Atari computers and software that are indispensable to both beginning and advanced Atari owners. They are: *The Atari User's Encyclopedia*, *The Book of Atari Software* (updated yearly), and *Atari Graphics and Arcade Game Design* by Jeffrey Stanton and Dan Pinal.

The single most valuable source for color graphics was given to me by a friend, Raymond Wells. It is an article called "Computer Animation with Color Registers," by David Fox and Mitchell Waite, which appeared in the November, 1982 issue of *BYTE* magazine. The article contains a good chart for color in the different graphics modes.

Another source of help was the Atari organization. The Customer Service Department has eight articles to help you use your computer; one in particular teaches you about color on the Atari.

All the books and magazines I've mentioned have been very good. None of them claim to be *the* book or *the* magazine. Each has its strengths, as well as its weaknesses. Refer to them all and increase your mental library of computer knowledge.

At this time, I'd like to acknowledge my fourteen-year-old son, Vincent; his help at various times was tremendous. I'd like, also, to thank Vincent's friend, Steve Wells, for his assistance.

Additionally, I thank the Merced Union High School District, especially John Lenker, Gerald Orchard, and Pauline Storck, without whose help I'd still be preparing a manuscript to send to a potential publisher.

Introduction

I wrote this book because of the frustration I encountered as a beginner trying to learn to use my computer. I read many books and articles that taught me a little, but then the authors would switch gears and aim their programs/demonstrations at the intermediate/advanced programmer, leaving me feeling confused and awfully dumb. I could type in the programs, but I couldn't figure out how they worked.

This is a tutorial for the beginner. At every step, I asked myself, "Will the beginner understand this?" If there were any doubt, I rewrote the section or explained the concept in another way. Sometimes, I explain the same thing differently in different chapters. I promise that you will not find short, snappy programs that are eye-catching, but impossible to understand. You will learn to program and to use color and graphics on the 400, 800, or on one of the new XL machines. Atari was gracious enough to allow me to spend a day at their facilities in Sunnyvale, using a new 600 XL, so I know the programs in this book work on the XL models.

Hints

The best way to learn from this book is to read it while you sit in front of your computer and type in the programs. Therefore, this is more a *workbook* than anything else. In addition, spend as much time as you can at your computer, using the knowledge you've gained by designing your own programs.

Another hint is that you should save all or a number of these programs on cassette or disk. When you finish the workbook, run them and marvel at how simple they actually are, but how complicated they seemed when you first began.

Another good reason to save them is that later you will learn better and faster ways to present the programs and you can make time-saving changes in the early ones. They will comprise a kind of test for you. *Remember* to underline things that are interesting, complicated, or unclear. Make marginal notes to yourself about these points. For clarity's sake, we've set the programs apart in the text by indenting the program lines. When you're typing in the programs, please *do not* indent the lines.

One more hint is to make notes on the inside of the front cover. Why not list the page numbers on which new concepts are discussed or new techniques are demonstrated? You'll probably end up owning several books and it's worthwhile to compile an easily-accessible index of important page numbers for each one. So get comfortable, flex your fingers, and enjoy your Atari!

THEORY OF THE EARTH

The theory of the earth is a branch of geology which deals with the origin and development of the earth and its various parts. It is a science which seeks to explain the processes which have shaped the earth and its various parts. The theory of the earth is a branch of geology which deals with the origin and development of the earth and its various parts. It is a science which seeks to explain the processes which have shaped the earth and its various parts.

1. THE EARTH

The earth is a planet which is the third from the sun. It is a sphere which is covered by a thin layer of water and a thin layer of land. The earth is a planet which is the third from the sun. It is a sphere which is covered by a thin layer of water and a thin layer of land. The earth is a planet which is the third from the sun. It is a sphere which is covered by a thin layer of water and a thin layer of land.

EDITING




100



Editing

Before we get into programming, color, and graphics, I'd like to take a little time to talk about some special keys on your Atari keyboard. As you know, much of the keyboard is like a standard typewriter. A number of keys, however, are different. Let's talk about some of these; they are the ones we use for editing.




No matter how hard you try to type everything perfectly, you're going to make mistakes. Luckily, the editing capabilities of the Atari are superb. (You'll find this out if you ever use another computer.)

Turn on your computer, please. The white rectangle you see on the screen () is essential to the editing process. It is called the *cursor*. If you press one character key, that character will appear where the cursor is and the cursor will move one space to the right. The cursor lets you know where you are on the screen.

Let's look at the keys you use to move the cursor to make corrections in your programs.



CTRL This key is always used in conjunction with other keys. It's the ConTRol key. Press it and you see that it doesn't "do" anything.



The four keys that follow are called cursor control keys.


 This is the up arrow key. Hold down the **CTRL** key and press the  key once. You have moved the cursor which now sits on top of the "R" in READY. Hold down the CTRL key and press the  key twice. Your cursor is now on the bottom line of the screen. This comes in handy when you wish to move the cursor quickly from the top of the screen to the








bottom. Now press the CTRL key *and* the up arrow key simultaneously, and keep them both pressed down. The cursor, after a slight hesitation, quickly moves up. Most of the keys on the keyboard perform this repeating function.

 This is the down arrow key. Press the CTRL key and the  key at the same time. The cursor moves down one line. If you'll keep both keys depressed, the cursor will move down swiftly. If you hold both of the keys down long enough, the cursor will eventually arrive at the top of the screen.

 This is the right arrow key. Press down the CTRL key and the  key once. The cursor moves to the right. Press both keys and hold them both down. If you do so long enough, the cursor will arrive at the left of your screen again.

 This is the left arrow key. The cursor moves one space to the left. As with the other keys, you will activate the repeat function if you keep both keys depressed. Do so and notice the cursor will eventually come up on the right side of your screen.

TAB When you press the TAB key, the cursor moves a pre-set number of spaces to the right. Try it. Just press the TAB key seven or eight times. Notice that it has the same wrap-around feature that the , , , and , keys have. You now can see the value of the TAB key in editing. It will swiftly move the cursor to the vicinity of an error, then you can use the CTRL/arrow keys for "fine tuning."

Throughout the book, I use the slash mark (/) to indicate that you hold down the first key as you press the second. "CTRL/right arrow" means hold down the ConTRol key as you press the  key, for instance.


Let's use these keys and a few others to make some corrections. Type in this program:

```
10 PRINT "SNOW WITE" (When I use parentheses after a
                      program line, the words within them are instructions or
                      comments. Do not type in anything between the paren-
                      theses unless I specifically ask you to do so. Press the
                      RETURN key.)
```

Next, type RUN and press the RETURN key. When you press the RETURN key, you enter the program line in the computer's memory, or in this case, you tell the computer to execute the command, RUN.

Your screen should look like this:

```
10 PRINT "SNOW WITE"
RUN
SNOW WITE

READY

```



1 EDITING

Whatever you put between quotation marks, the computer prints (with a few exceptions that we won't go into here). It doesn't know if you have made a typing error and it doesn't care, since the error doesn't affect the execution of the program. Since you know and care, however, you will want to correct it. There are two general ways to accomplish this: you can erase the program and start over, or you can correct the error.

One way to correct a mistake is to type the word NEW, then press the RETURN key. When you do this, the computer erases the program, which in this instance is only one line long. You may then type in the line correctly.

Type NEW, then press the RETURN key. This should erase the program. To find out if it did, type LIST, then press the RETURN key. This command causes the computer to LIST (or make a list of) all program lines.

Your screen should look like this:

```
READY
10 PRINT "SNOW WITE"
RUN
'SNOW WITE
```

```
READY
NEW
```

```
READY
LIST
```

```
READY
□
```

After the command "LIST," the computer will place on the screen any program you have entered. Since you typed NEW before you typed LIST, the program was erased—so there is nothing to LIST. This is the first way to correct an error: simply erase the program and start over.

The first method is fine if your program is short, but you won't want to use it for long programs. Let's use the second method for the same kind of problem.

First, press the SYSTEM RESET key in the upper right-hand corner. This erases the screen, so you have:

```
READY
□
```

and a clear screen. Type this in, please:

```
10 PRINT "WALT DINSEY"
RUN
```

I assume that you always press the RETURN key at the end of a program line

or at the end of a command such as RUN or LIST. If you don't, the computer will not store the program in its memory and will not execute the command. Whenever I say RUN it, I mean that you should type in the word RUN, then press the RETURN key.

Your screen should look like this:

```
READY
10 PRINT "WALT DINSEY"
RUN
WALT DINSEY

READY
□
```

You now want to correct the spelling of this creative genius' name, but you don't want to erase the entire line and type it in again. Press the CTRL key, hold it down, and press the up arrow key until you reach the numbered line. At this point you may release these two keys and press the TAB key twice. Now use the CTRL/right arrow key to move the cursor to the letter "N" in DINSEY. All you have to do now is type the letters "SN" and press return. The spelling has been corrected!

These are the two methods you'll use to correct mistakes. The first is to erase the line and type it over. The second is to make the correction within the numbered line itself.

If your screen is the same as mine at this point, your cursor is on the "R" in RUN. If you press the RETURN key, the computer will execute whatever is on that line. Since the command RUN is there, the computer will again RUN your program. Press the RETURN key.

Your computer again printed the name. This time, though, it printed it correctly over the formerly incorrect spelling.

Another option you had was to press the SYSTEM RESET key when the cursor was on the "R" of RUN. As you know, that would have erased the screen, but not the program in memory. Then you could have LISTed the program or RUN it.

There is yet another option: press the SYSTEM RESET key, then RUN the program. Now move the cursor to the "R" in RUN. You know if you press the RETURN key, the computer will execute the program again. Instead, press the BREAK key four times. You are now on the line below the READY sign.

One use of the BREAK key is to avoid executing commands. As your programs become longer, you'll see the real value of using the BREAK key. Remember that immediately after you type a line or make a correction in a line, you must press the RETURN key once to record your line or correction in the computer's memory.

Let's use another one of the editing keys. Type NEW and press RETURN. As

1 EDITING

you recall, this will erase the program in the computer's memory. Type in this program, please:

```
10 PRINT "DONALD DUK"  
RUN
```

Gosh, another mistake. Let's correct it a new way.

Move your cursor to the numbered line. Use the TAB key and the CTRL/right arrow or /left arrow keys until the cursor is on the "K". Then press and hold down the CTRL key and press the INSERT key once. When you use the CTRL/INSERT keys, the computer will move the letter that is directly under the cursor (and all letters to the right of it) one space to the right. You now have:

```
10 PRINT "DONALD DU[K]"
```

The cursor is now in place. Please type in the "C" and press RETURN twice. Everything's fine, at last. When you press RETURN once, you record your correction; pressing it again RUNs the program and the Atari prints the corrected version of Donald's name.

Look now at another key used for editing. Type NEW, press RETURN, then type in this program:

```
10 PRINT "MICKEY MOOUSE"  
RUN
```

Let's correct Mickey's last name. I'm sure you recall that any changes must be made in the *numbered* line.

Move your cursor to the first or second "O" in the word MOOUSE. Press the CTRL key, hold it down, and press the DELETE key once. This erases the "O" under the cursor and the text to the right of the cursor moves one space to the left. Press RETURN. Now press the BREAK key four or more times, type RUN, press RETURN, and Mickey's last name is spelled correctly. Pressing the BREAK key moves the cursor down one line. Pressing it four times moves the cursor down four lines without entering the lines or executing any commands.

Another use for the CTRL/DELETE keys is to erase an unnecessary space. Use NEW to clear the computer of a previous program, and type in the following program, please:

```
10 PRINT "HOWDY (press the SPACE BAR twice) DOODY"  
RUN
```

To eliminate one of the spaces between the two names, move the cursor to one of the spaces. Now press the CTRL/DELETE keys. Everything to the right of the cursor moves one space left. In a sense, the cursor gobbles up one of the spaces—the original Pac-Man.

If, at this point, you press the BREAK key, the cursor will move down one line,

but the correction will not be entered into the computer's memory. Also, if you use the CTRL/up or CTRL/down arrows, the program will not be corrected. The same is true if you use the SYSTEM RESET key; you must press the RETURN key to record the corrected line.

Press RETURN, then press the BREAK key four times and RUN it. The extra space has been DELETED.

Let's look at another kind of error. Use NEW to clear your computer of old program lines and type in the following program:

```
10 PRINT !ANOTHER KIND OF MISTAKE."
```

As soon as you pressed RETURN, you were in hot water. Your screen should look like this:

```
10 PRINT !ANOTHER KIND OF MISTAKE."  
10 ERROR- [ ] PRINT ANOTHER KIND OF MISTAKE."
```

Another cursor is on the exclamation point, showing you where you made the mistake: you should have used a quotation mark instead of an exclamation point. (Don't be fooled. The cursor is *not* always at the spot where you made the mistake.)

There are, as you know, a number of ways to make the correction. As a rule, it's easiest and best to use the following approach.

Move your cursor to the one in the *second* line 10. Press the SHIFT key, hold it down, and press the DELETE key. This erases that line from the screen, but it does not correct the error. Next, move your cursor to the exclamation point in the line 10 that remains, type the quotation mark over it, and press RETURN. Now RUN it. All is well.

We've talked so far about two kinds of errors and how to correct them. The first kind of error is one *you* notice occurring between quotation marks. The second is one the *compute* notices immediately after you type in a line and press RETURN. Let's talk about the third general error and how to correct it.

If you look ahead a bit, you'll see there is a program coming up. I won't explain how the program works. We'll be going over many programs, line-by-line. For now, we'll assume you understand everything in the program in order to illustrate the third kind of error, the type that occurs when you try to RUN a program.

Press SYSTEM RESET, type in NEW, and press RETURN. Then type in the program, please. Remember to press RETURN at the end of each numbered line.

```
10 GRAPHICS 3+16  
20 COLOR 1  
30 PLOT 2,2  
40 DRAWTO 37,2
```


1 EDITING

```
50 FOR X=1 TO 500:NEXT J
60 COLOR 3
70 PLOT 20,2
80 DRAWTO 20,33
90 GOTO 90
RUN
```

Since your computer executes the program so swiftly you may not have seen it, but the computer drew a line across the top of the screen. Then it issued the error message:

```
ERROR- 13 AT LINE 50
```

If you are a beginner, the error message won't help very much. Get out your *Atari BASIC Reference Manual* and look at Appendix B; it contains an explanation of various error messages. The explanation for ERROR 13 is that there is no matching FOR statement. That probably doesn't make much sense to you right now, but don't worry, it will in time. For now, let me tell you how to correct this error without going into the reasons why.

After your computer has printed an error message, you may RUN or LIST your program. Please LIST it (type LIST and press RETURN). Now place your cursor over the J in line 50. Remember to use the CTRL/right arrow keys. Now type an X over the J and press RETURN. Next use the SYSTEM RESET key and RUN the program, or press the BREAK key until you are on a clear line and RUN it.

You fixed that problem because the computer drew a line across the top of the screen, paused, then drew a line from the middle of that line to the bottom of the screen. Unfortunately, you got another error message:

```
ERROR- 141 AT LINE 80
```

Looking again in Appendix B, you'll see the error involves a cursor problem. If you don't know what this error means, let's assume you do.

LIST your program again. Place the cursor on the first 3 in line 80. Type in a 2 and press RETURN. Press the SYSTEM RESET key and RUN it, or use the BREAK key until you have a clear line, then RUN it.

Great! The program works. You now have a capital T in two colors.

Because of line 90, the computer will continue to display the T. When you are ready to return to the programming screen, press the BREAK key or the SYSTEM RESET key. If you wish, you may now LIST the program or get rid of it.

Let's erase the program and put in a different one to demonstrate another kind of error and correction method. Type NEW, press RETURN, then enter this program, please:

```
10 ? "MY ADDRESS IS 12 EAST FOURTH STREET,"
```



```
20 ? "MY NAME IS JAMES ALLEN."  
30 ? "I AM 45 YEARS OLD."  
40 ? "ALBUQUERQUE, NEW MEXICO"  
RUN
```

It comes out okay, but not in the order in which you want it. You could erase the program (by typing NEW and pressing RETURN), then change the information in each line and re-type it. Or, you could go to each line and correct whatever you'd like between the quotation marks.

Let's do it a bit differently. Type the number 10, then press RETURN. Type the number 20, then press RETURN and type the number 30, then press RETURN.

You have just erased these three lines from the program. Now move your cursor to the 1 in line 10 of the program. Type a 3, and press RETURN.

Your cursor is now on the 2 in line 20. Type a 1, and press RETURN. Your cursor is now on the 3. Type a 2, and press RETURN. Press the BREAK key six times. The line you are on is between the word "ALBUQUERQUE" and the word "READY." RUN it. Now the program is better organized.

By typing in the numbers by themselves and pressing RETURN, you erased those numbered lines from memory. (Whenever you have program lines out of order, you can use this technique. You'll be surprised how many times you'll make corrections this way.)

When you moved your cursor to the erased lines, changed the numbers, and pressed RETURN, you recorded the lines again, but with the correct line numbers. When you ran it, the information on this person was displayed in the proper sequence.

As you type in these programs, you'll probably make errors. Most of them will be typing errors. With the information you've learned in this chapter, you should be able to fix most mistakes you encounter. Appendix B of the *Atari BASIC Reference Manual* will be increasingly helpful as you progress through the book and learn more computer terms.

If you're not feeling totally comfortable correcting errors, you should re-read this chapter and type in the exercises again. But if you're feeling comfortable with the keyboard, let's go on to the next chapter.

GRAPHICS MODES



1.1. Introduction



Graphics Modes

The word “graphic” comes from the Greek word, *graphein*, which means “to write.” In the computer world, graphics refers to whatever is “written” (or displayed) on the TV screen or monitor, whether colors, letters, numbers, words, symbols, or drawings.

The word “mode” comes from the Latin word, *modus*, meaning “measure or manner.” When computer users use it, we mean a form or variety of screen display.

When computerists use the term “graphics modes,” we are referring to different varieties of screens, or grids, on the TV or monitor. One graphics mode may print large characters; another may print small characters; a third may display drawings only.

The Atari has a number of graphics modes, depending on the kind of computer you have. The early Ataris have a computer chip called the CTIA. This chip supports sixteen modes, but only nine of them are easily used by the beginner: GRAPHICS 0 through GRAPHICS 8.

Recent Ataris come with a chip called the GTIA. This new chip allows the beginner to use GRAPHICS 0 through GRAPHICS 11. Again, other graphics modes are available, but knowledge of advanced programming techniques are necessary to use them.

Most recently, Atari has manufactured the XL line. With these new computers, the beginner can use GRAPHICS 0 through GRAPHICS 11, GRAPHICS 14, and GRAPHICS 15. In this book you will learn to use all of these modes. GRAPHICS 12 and GRAPHICS 13 are available, but for advanced programmers. Since this is a beginner’s tutorial, these two modes will not be covered.

2 GRAPHICS MODE

To better understand what a graphics mode is, let's examine the following grid:

	0 COLUMN	1 COLUMN	2 COLUMN	3 COLUMN	4 COLUMN	5 COLUMN	6 COLUMN	7 COLUMN
ROW 0								
ROW 1								
ROW 2								
ROW 3			R					
ROW 4								
ROW 5								
ROW 6								
ROW 7								
ROW 8								
ROW 9								
ROW 10								

This grid is composed of a number of "invisible boxes." The boxes—stacked one on top of another—are called columns. The boxes, placed side by side, are called rows. To make matters very simple, let's say that a graphics mode is really an invisible grid made up of a number of boxes arranged in columns and rows.

The invisible grid is placed on the TV or monitor screen. There is, however, some space above and below the grid, as well as to the right and left of it in most modes. This space is called the border, and it is put there by your computer because some TVs and monitors display their screens in different fashions. By setting aside a border like this, Atari insures that whatever you put inside the

invisible grid will be displayed on your screen. If your computer did not set aside this border, part of your display might be eliminated on some screens.

This invisible grid has invisible numbers. The columns are numbered from left to right, starting with the number 0. The rows are numbered from top to bottom, beginning with the number 0. In our illustration of a grid, the columns are numbered from 0 to 7 and the rows are numbered from 0 to 10. (Numbering on your Atari starts with 0.)

To continue my simplification of graphics modes, your computer has several different invisible grids. Some contain a small number of large boxes and others contain a large number of small boxes. These grids are called graphics modes. You choose one of these grids whenever you type in the word **GRAPHICS** and follow it with a number. As we mentioned earlier, different models of the Atari have a number of modes that are easy to use.

After choosing a mode, you use this invisible grid to write or draw on the screen. If you want to write on the screen, choose **GR.0**, **GR.1**, or **GR.2** (**GR.** is the abbreviation your computer accepts for graphics). These are the three text (or writing) modes. To place the name "ROGER" on the screen, you have to tell the computer where to place it. If you don't, your computer will place it in the upper left-hand corner of the screen, or after the word "RUN."

On the preceding grid you'll see the letter "R" at the intersection of column 2 and row 3. If you want to use that spot for the "R" in "ROGER," you have to type **POSITION** (or **POS.**) 2,3. When you "position" something, you use the column number first, followed by the row number. This tells the computer where to print something. **POSITION** is the Atari BASIC command that locates the cursor at a given position on the screen and can be used in all graphics modes.

The following program shows you how to print the name "ROGER" on a **GR.0** screen beginning at column 2, row 3. Type in the program, please:

```
10 GRAPHICS 0
20 POSITION 2,3
30 ? "ROGER"
40 GOTO 40
RUN
```

Did you remember to press **RETURN** after each line and to press **RETURN** after you typed **RUN**?

You have printed the name beginning at column 2, row 3. Since **GR.0**, unlike our "visible" grid, has forty columns and twenty-four rows, the name is in a different spot—the upper left-hand quadrant.

If you use **GR.1** or **GR.2**, you have to use **PRINT#6**; instead of **PRINT** to accomplish the same thing. In any graphics mode, however, you must always use a set of quotation marks (" ") around the word, or words, you want to print. The

2 GRAPHICS MODE

PRINT command is reserved for the GRAPHICS 0 screen and the GRAPHICS 0 text window (the text window is covered in a separate chapter). The PRINT#6; command is usually reserved for the screen portion of the GRAPHICS 1 and GRAPHICS 2 modes.

Type in the following program. Before you do, though, press SYSTEM RESET, type in the word NEW, and press RETURN.

```
10 GR.2+16 (Don't worry about the +16 for now.)
20 POS.2,3
30 PRINT#6;"ROGER"
40 GOTO 40
RUN
```

You have the name **ROGER** printed in green on the GR.2+16 invisible grid beginning at POS.2,3. Since this mode has only 20 columns and 12 rows, the name does not appear in the same spot on your screen. It is, however, still at POS.2,3. Notice that the name is much larger in GR.2+16 than it was in GR.0. We'll discuss the different sizes as we discuss each mode in depth.

Let's summarize what you've learned so far about graphics modes:

- 1) Your Atari has a number of invisible grids that can be placed on your screen.
- 2) The grids vary in the number and size of the "boxes" they contain. These grids are actually modes.
- 3) In GR.0, use POSITION (or POS.) and PRINT statements to place characters on the screen at a certain location.
- 4) In GR.1 and GR.2, use POSITION (or POS.) and PRINT#6; statements to place characters on a screen at a pre-determined location.
- 5) In GR.0, GR.1, and GR.2, you surround the word (or words) you wish to display by quotation marks.
- 6) The size of the word(s) displayed will be different in different graphics modes.

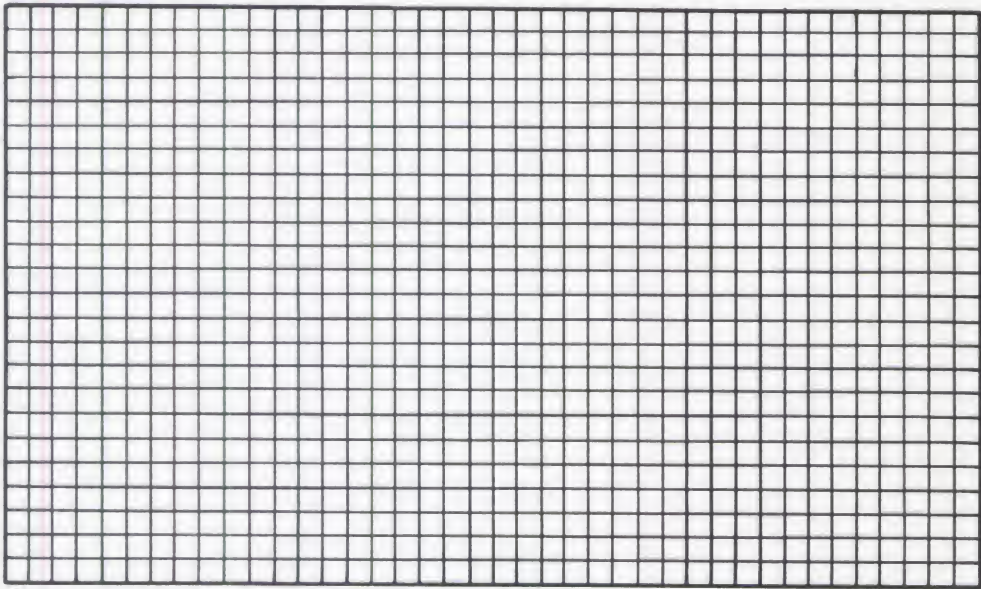
As you can see, it's not all that difficult. But before we leave this quick overview, let's talk about the non-text, or drawing modes. The grids, again, vary as to size and the number of boxes; in this respect, the text and drawing modes are similar. One major difference is that you must tell the computer what color to use in order to put anything on the screen in a drawing mode. Since your Atari is probably the best color machine on the personal computer market, you can expect that it's a bit difficult to use color. The rest of the book is devoted to the use of color—in the different graphics modes and in your programming. For now, we'll confine ourselves to the approach used to draw a colored line on the grid/screen. Please type in the following program:

```
10 GRAPHICS 5+16
20 COLOR 2
```



```
30 PLOT 4,3
40 DRAWTO 34,19
50 GOTO 50
RUN
```

Your computer drew a green diagonal line from the intersection of column 4, row 3, to the intersection of column 34, row 19. In any mode, when giving the coordinates of the spot you want, always put the column number first, followed by the row number. Again, the numbering on the grids—whether they are for text or drawing—always starts with 0 in the upper left-hand corner for the columns, and 0 in the upper left-hand corner for the rows.



40 x 24 Grid GR.0

Typing in the word “PLOT” causes the computer to place a dot of color in the location 4,3. The word “DRAWTO” tells the computer to DRAW a line of color from the PLOT location TO location 34, 19.

Essentially, the text and drawing modes do different things using different-sized grids.

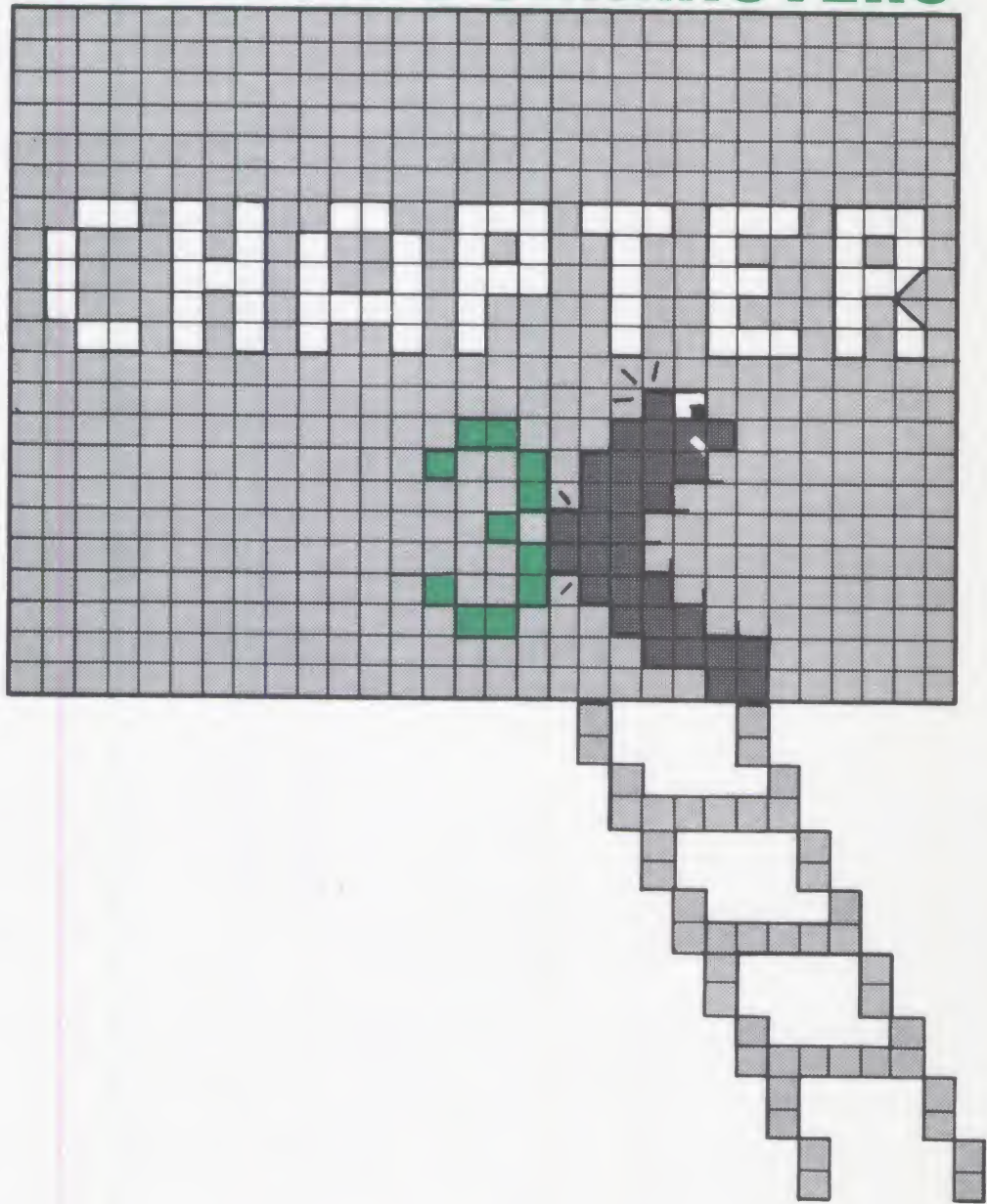
This chapter has explained how your computer uses grids to place characters, color, and lines on your screen. We’ll delve deeper into the “hows” beginning with the next chapter. If you’re a bit anxious, don’t worry. I’ll go very slowly and explain everything carefully and completely. I remember my own green and frustrating days and sympathize with you. Just relax and enjoy acquiring new computing skills.

The first part of the paper discusses the importance of the research and the objectives of the study. It also provides a brief overview of the literature review and the methodology used in the study.



The second part of the paper presents the results of the study and discusses the implications of the findings. It also includes a conclusion and a list of references.

DISPLAYING CHARACTERS





Displaying Characters

There are two main concepts you will learn from this chapter. The first is how your computer places a character on your screen; the second is why characters can be different sizes.

You've already learned that a graphics mode has a certain number of boxes in a grid. GR.0, for example, has a grid that is forty boxes wide (columns) and twenty-four boxes high (rows). GR.0, then, has 960 boxes you may "fill."

How does your Atari fill these boxes with letters, numbers, and symbols? It's easy. Your computer builds another grid and places it inside one of the boxes already on the screen.

This grid is composed of sixty-four "light bulbs" which are placed inside one box of your screen grid. There are eight columns of these light bulbs in eight rows, totalling sixty-four.

Your computer uses a grid inside a grid. It takes one of the 960 boxes in GR.0 and divides it into sixty-four tiny boxes. Then it places a tiny light bulb inside each of these boxes. Since you have sixty-four light bulbs inside each of the 960 boxes in GR.0, you have a total of 960×64 , or 61,440 light bulbs on your screen. Each one of these light bulbs can be "on" or "off."

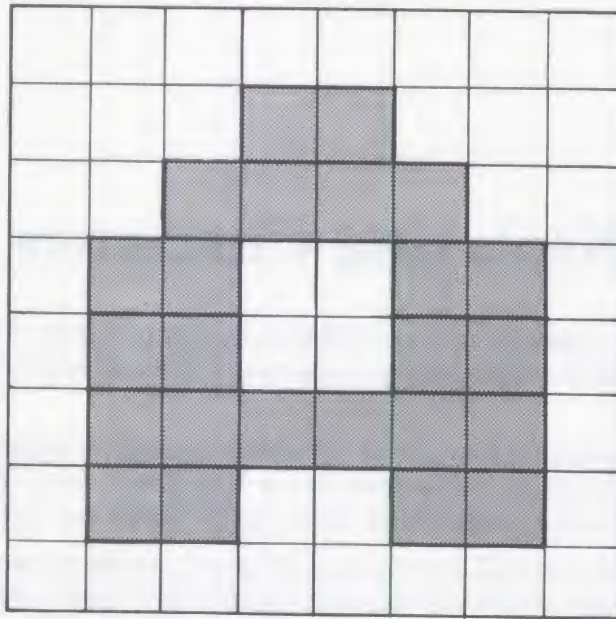
What follows is an explanation of how your computer places the letter "A" on your screen.

A part of your computer addresses each one of these sixty-four light bulbs. It turns some on and some off. According to the pattern it chooses, a character is displayed on the screen. Each "A" for example, is not really what it seems to be. It is actually sixty-four light bulbs, some on and some off, producing specks of light that we interpret as the letter "A." You can observe this on marquees where light bulbs are turned on in such a fashion that they represent a letter. That's the same

3 DISPLAYING CHARACTERS

thing your computer does (with the help of your TV) inside each one of the 960 boxes in a GR.0 grid.

Perhaps things will be clearer if we look at how the computer places an **A** on the screen.



ONE OF THE 960 BOXES IN GRAPHICS 0 (GR.0)

When you press the A key, your computer turns on twenty-four light bulbs in the shape of an **A**.

Notice, first, that the computer turns off all the bulbs on the edges of the box. It doesn't illuminate them because it wants to leave a space between each character (just as there is a space between these two characters: A B). If it did not turn off these lights, characters would touch each other at the sides, tops, and bottoms. Secondly, it turns on lights in a pre-determined fashion, so the letter **A**, or a comma (,), or an **M** appears.

Each of these sixty-four light bulbs is called a *pixel*. It takes one *bit* to turn on a pixel. A line or row of eight bits is called a *byte*. In other words, it takes eight bits to make one byte and it takes eight bytes to store the instructions that form a character or a symbol on the screen. It is the illumination—or non-illumination—of the sixty-four bits that produces a character or symbol on your screen in graphics mode.

This entire chapter gives you a beginning knowledge of the process your computer uses to place letters, symbols, and dots of color on your screen. As you may guess, the subject is much more complicated. If you wish to pursue it further, research the raster scan method of display. There is quite a bit more to bits, bytes, and pixels than what I have presented, but with the understanding you now have, you can comprehend the method easily.

Now that you've learned how your computer places symbols and characters on your screen, you're ready for the second concept: how the characters can be different sizes.

We've gone over several things that relate to understanding different-sized characters. Let's recapitulate them:

- 1) A graphics mode is composed of a set number of boxes in a grid.
- 2) There are a number of different graphics modes with different-sized grids.
- 3) To place a character on the screen, your computer divides *each* box in a grid into sixty-four smaller boxes; to place a character on the screen, it turns light bulbs on or off in a specific fashion.

Throughout this book, you'll be building on knowledge you learned earlier. With the above information under your belt, it will be easy to see how your Atari can print characters of different sizes. We've spent a lot of time with GR.0; let's look now at a new mode, GRAPHICS 1 (GR.1), and compare it to GR.0.

GR.0 has forty boxes in each row. Each box contains sixty-four light bulbs, so there are forty boxes with sixty-four light bulbs each, totalling 2,560 light bulbs in each row.

In GR.1, you have twenty boxes in each row. Each box contains sixty-four light bulbs for a total of 1,280 light bulbs, half the number of GR.0.

Since GR.0 and GR.1 fill the same area of the screen, the light bulbs in GR.1 will have to be wider than the ones in GR.0; in fact, they are twice as wide, meaning the characters will be twice as wide in GR.1. Both GR.0 and GR.1 have the same number of rows, though, so the characters are the same height. I'll bet you took the next leap. If we change the number of rows, we'll also change the height of the characters. Let's look at the size of the grids for three graphics modes:

GR.0	40 columns and 24 rows
GR.1+16	20 columns and 24 rows
GR.2+16	20 columns and 12 rows

For the time being, don't worry about the "+16." You'll learn about this later. You can see that GR.2+16 has only half the number of rows that GR.0 and GR.1 have. Consequently, each box in the grid has to be twice as tall in this mode to fill

3 DISPLAYING CHARACTERS

the same size screen as GR.0 and GR.1. Both the light bulbs and the characters are twice as tall. To summarize:

GR.0	40 columns and 24 rows	normal-sized characters.
GR.1+16	20 columns and 24 rows	twice-as-wide characters.
GR.2+16	20 columns and 12 rows	characters that are twice as wide and twice as tall as GR.0 characters.

The key phrase to understanding all this is "grids within grids." Since you will write in GR.0, GR.1, and GR.2, your characters will be different sizes. This is convenient because you won't always want your numbers and letters to be the same size. They can be different because of "grids within grids."

Up to this point, you have learned how your computer places characters on the screen and how you can make them different sizes. You're now ready to begin your study of the different graphics modes.

GRAPHICS ZERO



0035 80184718



Graphics Zero

GRAPHICS 0 is the mode in which you type your programs. When you turn on the computer, it is automatically in GR.0. The user cannot PLOT and DRAWTO in this mode. You can, however, use the special Atari graphics symbols (in the back of your *Atari BASIC Reference Manual*) to design pictures. Also, you can use color in this mode for the background, border, letters, and special symbols.

Let's look at the list of sixteen colors available on your Atari:

0	gray	8	blue
1	light orange	9	light blue
2	orange	10	turquoise
3	red-orange	11	green-blue
4	pink	12	green
5	purple	13	yellow-green
6	purple-blue	14	orange-green
7	blue	15	light orange

When the computer talks about columns, rows, graphics modes, and colors, it always starts numbering with zero (0).

There are different ways to use these colors on your Atari. They are somewhat complicated, but certainly not impossible to learn. Part of the reason for the complication is that you can do much more with color on the Atari than can be done on most other computers. Many computers have a set number of colors (sixteen or fewer). An Atari can choose from a palette of up to 256 colors, depending on the model you have.

4 GRAPHICS ZERO

That statement, though, is misleading. If you want to do a realistic drawing of a house and a barn, you don't really have 256 colors to choose from. Depending on the graphics mode you choose, you have a maximum of sixteen colors.

Color Registers

To place color on your screen, you must understand color registers. Your Atari has nine of them (four are used for player/missile graphics, a very complicated subject which will not be covered in this book). Five color registers remain for you to use—but you can't use all of the registers in all of the modes.

I'll make this as easy as possible. Pretend that a color register is a paint jar into which you will place a specific color of paint. You will then use that color register (jar) for drawing with that particular color. You've seen the list of sixteen colors available on your machine, so how can you possibly have more than sixteen to choose from and place in a jar? This is possible because the designers of the Atari did not want to limit you to those sixteen colors only. They arranged for you to be able to change the brightness or luminance of all the colors. In most modes, you may choose eight brightnesses or luminances for each color.

Most specifically, you may choose one of eight brightnesses for each color; that will give you sixteen colors times eight brightnesses—a total of 128 colors.

How do you do this? First, you choose a color register (jar), tell the computer to place a certain color of a certain brightness in it, and then start drawing.

It gets a bit complicated because certain registers are used for certain things. In GR.0, color register 4 controls the color of the border only. Color register 2 divides the jar in half from top to bottom. One half of the jar contains the color for the background, the screen on which you compose. The other half contains a lighter shade of that same color for the foreground, or whatever you place on the screen.

So, you have a register for the border and a register for the background which *automatically* chooses a lighter shade of that same color for the characters you place on the background. Register 2, then, does double duty.

The Atari designers give you an option for the foreground shade. They set aside another register so you can change the brightness (luminance) of the foreground characters. This is register 1. For now, though, let's get into the specifics of choosing the colors for registers 2 and 4 in GR.0. We'll talk later about register 1.

As we continue this discussion, remember that we are talking about color in GR.0 only. You do not choose color the same way in all the graphics modes.

There are three steps to choosing a color in GR.0.

Step 1

First, pick the register. If you want to change the color of the background, you must choose register 2. Do so by typing in SETCOLOR 2 or SE.2. When you do, the computer knows you are going to place a color here for the background.

Step 2

Pick a color from one of the sixteen. Do this by adding one of those numbers on our list to your earlier statement. If you pick light orange, use the number 15. Add this number to your earlier statement:

```
SETCOLOR 2,15
```

Note the comma separating the two numbers.

Tell the computer how dark or how bright you want that color. As we mentioned earlier, there are eight brightnesses (luminances) available. They are 0,2,4,6,8,10,12, or 14. The lower the number, the darker the shade of light orange; the higher the number, the lighter the shade of light orange. Let's pick the darkest shade of light orange by adding a 0 to the earlier statement. You now have:

```
SETCOLOR 2,15,0
```

2 = COLOR REGISTER for the background
15 = COLOR chosen for the register (light orange)
0 = BRIGHTNESS of the preceding color

Step 3

You now know the three steps involved in choosing color in GR.0. First, pick the color register; second, pick the color; third, pick the shade (brightness or luminance) of that color.

In the other graphics modes, you use the color registers, but not in exactly the same way (we'll come to this in succeeding chapters).

Turn on your computer, please. On my screen, the major portion is light blue. There is a strip of gray at the top and bottom. In this mode, the strip comprises the border.

Type in the following program, please:

```
10 GR,0    (This is the abbreviation for GRAPHICS.)
20 SE,2,15,0 (This is the abbreviation for SETCOLOR.)
RUN
```

The background portion of your screen is now a greenish-brown. This is the darkest shade of color 15. (Caution: colors will vary from TV to TV.)

LIST your program. Remember, you can type LIST and press the RETURN key. Or, you may type L., the abbreviation for LIST, and press RETURN. Also, you may press the BREAK key or SYSTEM RESET key and LIST your program.

4 GRAPHICS ZERO

Now move your cursor to the 0 at the end of line 20 and type in a 14, then press RETURN. RUN it. The brightest shade of color 15 is now on your screen.

LIST your program again and change the 14 to each of the following brightnesses: 0,2,4,6,8,10,12, and 14. RUN each one.

When you use BRIGHTNESS 10, you will no longer be able to see the "READY" sign and the cursor. This is because the luminance of the letters (foreground) is normally 10. If you use 10 for the background as well, the foreground and background will be the same color; you will see neither the "READY" nor the cursor. After you use BRIGHTNESS 10, press the SYSTEM RESET key and LIST your program. Change the 10 to another number, and the READY/□ will re-appear.

If you worked at it, you've now seen the eight shades of color 15. Since each color has eight shades, you have a choice of 8×16 , or 128 colors. (Later, I'll talk about 256 colors.)

Take a little time off here from the book to experiment. This is what I'd like you to do: change the last digit in line 20 to a 0. Then choose another color for the middle digit in the SETCOLOR statement, and use the eight shades for it. In other words, do something like this:

```
20 SETCOLOR 2,7,0
```

RUN the program, LIST it, change the brightness (last digit), choose another color, and repeat the process. When you feel comfortable with the process of choosing colors for the background, change SE.2 to SE.4, and work on the border colors the same way you did the background colors.

After doing this, put SE.2 and SE.4 in the program and change both of them:

```
10 GR.0  
20 SE.2,8,0 (background/foreground)  
30 SE.4,8,0 (border)
```

Then RUN them through all their colors and shades.

Here are several reminders and a few hints before you're on your own:

- 1) Press RETURN to enter a line or execute a command.
- 2) You may use the full word or the abbreviations for GRAPHICS, SETCOLOR, and LIST: GR., SE., L. Be sure to include a period if you use an abbreviation.
- 3) If your program gets fouled up, correct it by using the different ways you learned in the chapter on editing.
- 4) If you delay changing colors, your Atari will do it for you. (It changes the colors automatically to keep from burning a line on your screen as early game machines used to do.) Pressing the BREAK key or any character key will return the screen to the color you chose.

- 5) You can make the color of the border and the background the same by using the same color and brightness in the two statements.
- 6) You can use opposite ends of a certain color spectrum for contrast:
SE.4,4,0
SE.2,4,14
- 7) SE.2 chooses the background and foreground, and SE.4 the border colors.
- 8) Keep at it until you're comfortable with changing color.

Okay, you're back and feel secure using color for the border and the background. Are you ready to change the color of a word? Please type in this program:

```
10 GR.0
20 SE.4,5,6
```

The number 4 above tells the computer you want to change the color of the border. The number 5 tells the computer you want the border to be purple. The number 6 controls the brightness of the color so it turns the purple to pink. It's the combination of these last two numbers that determines the actual color displayed.

```
30 SE.2,12,6
```

The number 2 tells the computer you want to change the color of the background. Number 12 chooses a green color. Number 6 makes it a light green. This background color chooses the printing color as well, but you can change the brightness of the color of the character you print.

```
40 SE.1,0,2
```

First, you can type the abbreviation SE. for the word SETCOLOR. Secondly, SETCOLOR 1,0 tells the computer you want to control the brightness of the letters, numbers, or symbols you place on the screen. You should put in the zero to indicate you are concerned only with the brightness of the color. The lower the number, the darker the color. Recall also that there are eight settings for the brightness: 0,2,4,6,8,10,12, or 14.

```
50 POKE 752,1
```

This deletes, or gets rid of, the cursor.

```
60 POS.14,11
```

This tells the computer that you want to put something on the screen at column 14, row 11. Remember, if you don't POSITION (or POS.) whatever you tell the computer to print, it will be displayed in the upper left-hand corner of the screen.

```
70 PRINT "PROGRAMMING"
```

This tells the computer what to display. The abbreviation for PRINT is the question mark (?).

4 GRAPHICS ZERO

80 GOTO 80



This tells the computer to stay on line 80. Thus the computer will leave the display on the screen. If you leave out this line, the computer would default to GR.0 after the program was RUN. GR.0 is also called the default mode because the computer returns to it after a program, unless there is an instruction such as line 80.

RUN

You see a pink border (line 20), with the upper case word **PROGRAMMING** (line 70), in a dark shade of green (line 40), at POSITION 14,11 (line 60). The background should be a lighter shade of green (line 30). To change the color of the word, change the last digit in line 40. You have eight choices: 0,2,4,6,8,10,12, or 14. *Remember*, SETCOLOR 1,0,# will change the brightness of the character you display on the screen in GR.0. If you wish, practice changing the color of the word.

LIST your program. Now we'll look at different ways to display the word. Move your cursor to the "P" in "PROGRAMMING." Press the CAPS LOWR key, release it, then type in PROGRAMMING again. You've typed the word again, but this time in lower case letters. Now hold down the SHIFT key and press the CAPS LOWR key simultaneously. You have returned your computer to upper case letters which you *must* use for programming and commands.

Press the RETURN key and RUN the program. You should see the word **programming**, in lower case.

Let's look at another way to display this word. Press the SYSTEM RESET key. LIST your program. Move your cursor to the "P" again, then press the Atari logo () key, or the DOM key. Release it, and type the word "programming" again. (XL owners have what is called a semaphore [or an inverse video] key which is half black and half white. When I asked you to press the logo or the DOM key, XL owners should press the semaphore [] key.) This will print text in inverse video. To return to normal text, press the DOM key again.

Press the DOM key now, then press the RETURN key. Afterward, RUN your program. You should see the word **PROGRAMMING** in capital letters with a dark background. This is an example of printing in inverse video.

LIST your program again, move the cursor to the "P," press the DOM key, release it, then press the CAPS LOWR key and release it. Now type "programming" again. Next, press and release the DOM key, then press the SHIFT key as you press the CAPS LOWR key. This returns your computer to normal printing.

RUN the program; you should have a dark background behind the word which is in light, lower case letters. You now have a lower case, inverse field word.

To sum up: when you type in lower case letters, you get lower case letters. When you use the DOM, semaphore, or logo key and uppercase letters, you get

inverse upper case letters. When you use the DOM key and lower case letters, you get inverse lower case letters.

To continue our summary, the foreground color is actually picked when you choose the background color because the computer will automatically put the foreground in a shade that's different from the background color. Thus, you will have a dark background with a light shade of the same color for your letters, numbers, or symbols. Recall, also, that you can change the brightness of the foreground with the SETCOLOR 1,0,BRIGHTNESS (0 to 14, even numbers only).

The number 6 will not work as a brightness in SE.1,0 in this particular program because it is the same intensity as the background. You may not use the same brightness in SE.1,0 that you use in SE.2.

Control Characters in GRAPHICS 0

Type NEW, and press the RETURN key. You have now erased the old program and are ready for a new one.

Many of the keys on your computer keyboard have graphics characters you can use. (Refer to the picture of your keyboard with these special characters on the keys in your *Reference Manual*.) Use these characters by pressing the CTRL key at the same time you press a letter key which has a graphics character beneath it. For example, if you hold down the CTRL key while you press the H key, your screen will display half a pyramid. Do so right now. Then hold down the CTRL key as you press the J key. You now have the other half of the pyramid.

Please type in this program:

```
10 GR.0
20 SE.1,0,14
30 SE.2,4,4
40 SE.4,9,2
50 POKE 752,1
```

Can you tell what each of the preceding lines will do? If not, go back to our earlier program and re-read the information.

The ? coming up is the abbreviation for the command PRINT. Whatever words or symbols you want to print *must* be within quotation marks.

You should be aware of an "eccentric" spacing habit practiced by the Atari when programs are LISTed. While typing in a program line, you may use a question mark (?) to abbreviate PRINT. When the program is subsequently LISTed, your Atari will *automatically* insert a space after the ?. The space will usually precede opening quotation marks. If, instead, you decide to type in the full word PRINT, the automatic spacing will not occur.

100 ?" (Hold down the CTRL key at the same time you press one of the character keys. In this case, hold down the CTRL key at the same time you press the) QWE" (keys.)

4 GRAPHICS ZERO

On the rest of these lines, make certain you put the ? followed immediately by the ". You should have:

```
100? "┐┐"
```

```
120 ?" (hold down the CTRL key) ASD"
```

Don't type anything inside the parentheses () unless you are asked to do so. You should have:

```
120 ?"┐┐"
```

```
140 ?" (hold down the CTRL key) ZXC"
```

You should have:

```
140 ?"┐┐"
```

```
RUN
```

You should have a window in the upper left-hand corner of your screen. But if you don't want it there, use the POSITION statement. Press SYSTEM RESET, LIST your program, and add the following lines. Type them in on clear lines after your program is LISTed and your computer will put them in their proper numerical order.

```
90 POSITION 7,7
110 POSITION 7,8
130 POSITION 7,9
RUN
```

These three lines told the computer where to place the CTRL/QWE, CTRL/ASD, and the CTRL/ZXC.

If you want to, you can make the window longer. Add the following lines to the program. Remember to press SYSTEM RESET or BREAK, and LIST your program. Now you will see why we number lines by tens.

```
125 POS.7,9
127 ?" (CTRL) ASD"
```

Now change line 130 to the following:

```
130 POS.7,10
RUN
```

The complete LISTing follows:

```
10 GRAPHICS 0
20 SETCOLOR 1,0,14
30 SETCOLOR 2,4,4
40 SETCOLOR 4,9,2
```



```
50 POKE 752,1
90 POS.7,7
100 ?"(CTRL)QWE"
110 POS.7,8
120 ?"(CTRL)ASD"
125 POS.7,9
127 ?"(CTRL)ASD"
130 POS.7,10
140 ?"(CTRL)ZXC"
RUN
```

From now on, I'll put CTRL/ASD, or whatever else is necessary, inside the quotation marks. You must press the CTRL key along with the other key or keys.

The word "READY" is still on the screen, isn't it? Type in the following and it will disappear:

```
150 GOTO 150
RUN
```

Now the prompt or "READY" sign is gone. But why? When your computer reaches line 150, it is told to GOTO line 150. It does so and is told to GOTO 150 again. You have a repeating cycle that stays on line 150, so the program never ends. The computer will not print "READY" because it isn't; it's stuck in a repeating cycle.

Erase the present program. Use the SYSTEM/RESET key, type NEW, then press the RETURN key.

Before you type in another program, I should explain something to you.

When you have a large number of lines that will not fit on one screen, your Atari will scroll the lines upward. You could, for instance, end up with line 660 at the top of your screen. This is unacceptable if you want to correct line 50. To interrupt the upward scrolling, do the following.

First of all, type L., press RETURN, and your computer will LIST the 660 lines. When you want to stop the LISTing, press the CTRL key and the number one (1) key simultaneously. The scrolling will stop. Assume that everything on the screen is okay and you have still more lines to LIST. Press CTRL/1 again and the LISTing will continue. Stop it once again by pressing CTRL/1. If there's a mistake in one of the lines, do the following:

Press the BREAK key. (This interrupts the LISTing by returning control of the keyboard to you.) Then use the editing keys to make the corrections. *Remember*—after you make the corrections, you *must* press RETURN.

Let's assume that you have LISTed a long program, interrupted it several times by pressing CTRL/1, used the BREAK key, corrected a line, and pressed RETURN. At this point, you may press the BREAK key continuously until you have a clear line, then type RUN or LIST again.

4 GRAPHICS ZERO

There are two other ways to LIST and correct. If you know the line number in which you have an error, type L.120 (or whatever) and line 120 will be LISTed. Make your corrections accordingly with the editing keys.

Another method is convenient when there is an error between line "A" and line "B." For example, if you think the error occurs between line 80 and line 190, type L.80,190 and your computer will LIST lines 80 through 190. If the error isn't there, type L.200,300, and the computer will LIST those lines.

REVIEW

To recapitulate, there are three ways to make corrections in a long program:

- 1) LIST your program; while it's scrolling, interrupt it with the CTRL/C keys. Press BREAK when you find an error; correct it.
- 2) LIST one specific line and correct it.
- 3) LIST a number of lines and see where you may have made an error.

Now you're ready to type in the following program, LIST it, and make any necessary corrections:

```
10 GR.0
20 POKE 752,1
30 SE.4,0,0
40 SE.2,12,2
50 SE.1,0,14
```

You can see from this that the SETCOLOR statements do not have to be in a specific order.

Now we are going to use the special characters on your keyboard to draw a house. When you draw a picture this way, you should start your drawing on a *three-digit line number* so the picture you get on the screen will look the same as the picture in your LISTing. It will also be easier for you to see any mistakes as you type. If you were to use double- and triple-digit line numbers, everything in the double-digit line numbers will be one space to the left of what is in the triple-digit line numbers.

```
100 ?" (press the SPACE BAR twenty-seven times. Press the DOM
key and then the SPACE BAR once. Press the DOM key again and
put in the final quotation mark.) "
```

(XL owners: remember the DOM key is your  key.)

```
110 ?" (press the SPACE BAR twenty-six times and press DOM once
and release it. Then press the SPACE BAR three times, then DOM
again.) "
```

```
120 ?" (type everything the same way as in line 110.) "
```

```
130 ?" (two spaces, then press CTRL/N twenty-four times. Re-
member, CTRL/N means hold down the CTRL key at the same
```

time you press the N key. Next press DOM and three spaces, then DOM again. You don't have to keep the DOM key depressed as you press the SPACE BAR three times.) "

140 ?" (two spaces, then CTRL/V. Twenty-five spaces, then CTRL/B) "

Now comes the handy trick you saw earlier in the chapter on editing. Move your cursor to the 4 in line 140. Type a 5 there, then press RETURN. Now move the cursor to the 5, type a 6 over it and press RETURN. Continue this process until you've typed in a 7, 8, and 9 and have pressed RETURN each time.

LIST your program. Neat, huh? You've typed line 140 six times with little effort. The trick is that when you typed in the new digit (or digits), you didn't erase the old ones so both lines are printed. You erase the old line by typing that number on a blank line and pressing RETURN. Since you didn't erase the old line number first, you kept it in the program and added the new line with the same information that was stored in the old one. I hope you didn't find this too complicated.

210 ?" (Two spaces, then CTRL/V. Two spaces and CTRL/QWE. Fifteen spaces and CTRL/QWE. Two spaces and CTRL/B) "

220 ?" (Two spaces then CTRL/V. Two spaces and CTRL/ASD. Fifteen spaces and CTRL/ASD. Two spaces and CTRL/B) "

230 ?" (Two spaces, then CTRL/V. Two spaces, then CTRL/ZXC. Eight spaces and CTRL/QWE. Four spaces and CTRL/ZXC. Two spaces and CTRL/B) "

240 ?" (Two spaces, then CTRL/V. Thirteen spaces and CTRL/ASD. Nine spaces and CTRL/B.) "

Now do the trick again. Put the cursor on the line 4 in line 240. Type in a 5 and press RETURN. Put the cursor on the 5 and type in a 6. PRESS RETURN.

270 ?" (Two spaces and CTRL/V. Thirteen spaces and CTRL/ZXC. Nine spaces and CTRL/B). "

280 ?" (Two spaces. Press DOM key; twenty-seven spaces. Press DOM again.) "

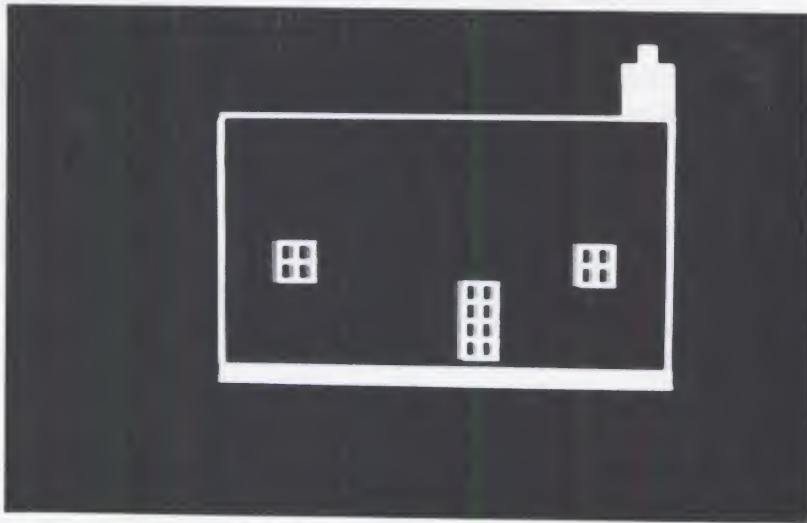
290 GOTO 290
RUN

You wind up with a picture of a house with a chimney, two windows, and a door. A Van Gogh it's not, but it shows what you can do with just a few control characters and the SPACE BAR.

If the sides of the house are out of place, the CTRL/INSERT and the CTRL/DELETE are marvels. It's difficult to give perfect directions when you're dealing with so many spaces and CTRL keys. If things don't look right, check

4 GRAPHICS ZERO

your line numbers against the ones in the book, then make corrections. Remember, the picture of the house should be the same as the picture in your LISTing. Be sure to SAVE any or all of the programs we do.



Type NEW and press RETURN. The next program will draw a face:

```
10 GRAPHICS 0
20 SETCOLOR 1,0,2
30 SETCOLOR 4,3,0
40 SETCOLOR 2,13,12
```

100 ?" (press DOM, then one space. Use the CTRL/right arrow keys to move twenty-three spaces. Press SPACE BAR once and DOM.) "

Remember the trick? Move the cursor to the first 0 in line 100, and change the 0 to a 1. Press RETURN. Repeat until you have typed lines 120,130,140, and 150, RETURNing each time.

```
160 ?" (DOM and one SPACE BAR. DOM again; three spaces.
CTRL/HJ and eleven spaces. CTRL/HJ; five spaces. DOM and one
space. DOM) "
```

Do the trick again. Move the cursor to the 5 in line 150. Make it a 7. RETURN. Move your cursor to the 7; make it an 8. RETURN. Move your cursor to the 8; make it a 9. RETURN and RETURN again.

```
200 ?" (DOM and one space. DOM and ten spaces. CTRL/SS;
eleven spaces. DOM and one space. DOM.) "
```

LIST your program. Do the trick again. Put your cursor on the 1 in line 180.

Change it to line 210. RETURN. Change line 210 to 220; RETURN. Change line 220 to 230; press RETURN until you are on a clear line. Remember, you must press the RETURN key to record the changes. You can LIST the program again and make certain that lines 210, 220, and 230 are the same.

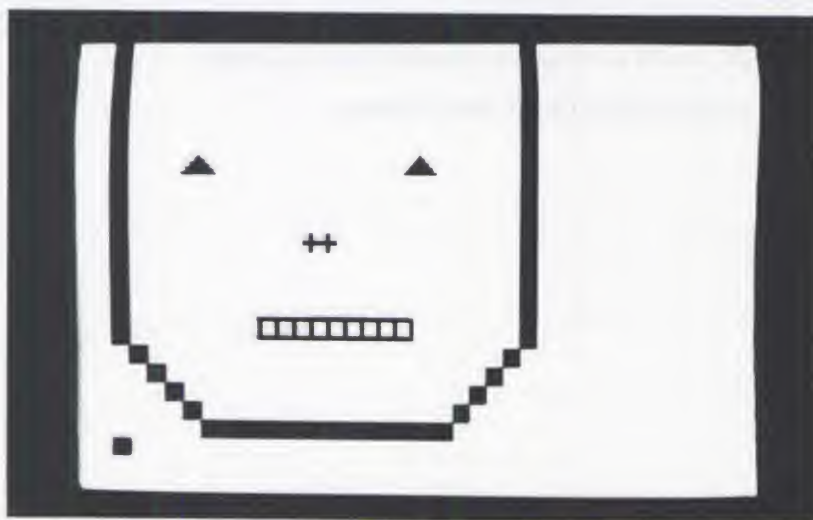
```

240 ?" (DOM and one space. DOM and seven spaces. CTRL/
QWWWWWWWE. [There should be eight Ws.] Six spaces. DOM
and one space. DOM) "
250 ?" (DOM and one space. DOM; seven spaces. CTRL/
ZXXXXXXXXXE. [There should be eight Xs.] Six spaces. DOM and
one space. DOM) "
260 ?" (One space. DOM and one space. Use the CTRL/right
arrow key to move twenty-one spaces. One space. DOM) "
270 ?" (Two spaces. DOM and one space. CTRL/right arrow
nineteen times. One space. DOM) "
280 ?" (Three spaces. DOM and one space. CTRL/right arrow
seventeen times. One space. DOM) "
290 ?" (Four spaces. DOM and one space. CTRL/right arrow
fifteen times. One space. DOM) "
300 ?" (Five spaces. DOM and fifteen spaces. DOM) "
310 GOTO 310
RUN

```





You have created a face using just a few graphics characters and the SPACE BAR.

This is how the face should look:



4 GRAPHICS ZERO

If your screen does not display a recognizable face, LIST the program. The face is there in your LISTing, too. Since you now know how it should look, use your editing keys. Remember to press CTRL/INSERT to move everything to the right; use CTRL/DELETE to move everything to the left. Be careful not to delete the PRINT statement or the quotation marks; if you do, you'll cause an error.

If your drawing of the face is a mess, don't worry about it. Many programs won't work perfectly the first time. Most problems will be typing errors. Sometimes the programmer's directions won't be clear (hopefully that wasn't the case here). View each problem as a challenge; leap in there armed with your CTRL key and the  ,  ,  ,  , INSERT, and DELETE keys and fix everything.

Now that you've corrected or "debugged" your program, experiment with the drawing. Center the face on the screen. Add ears to it. Change the symbols for the eyes, nose, and mouth. Change the colors. Make the head a triangle or a circle. The more you experiment, the more experienced you'll become.

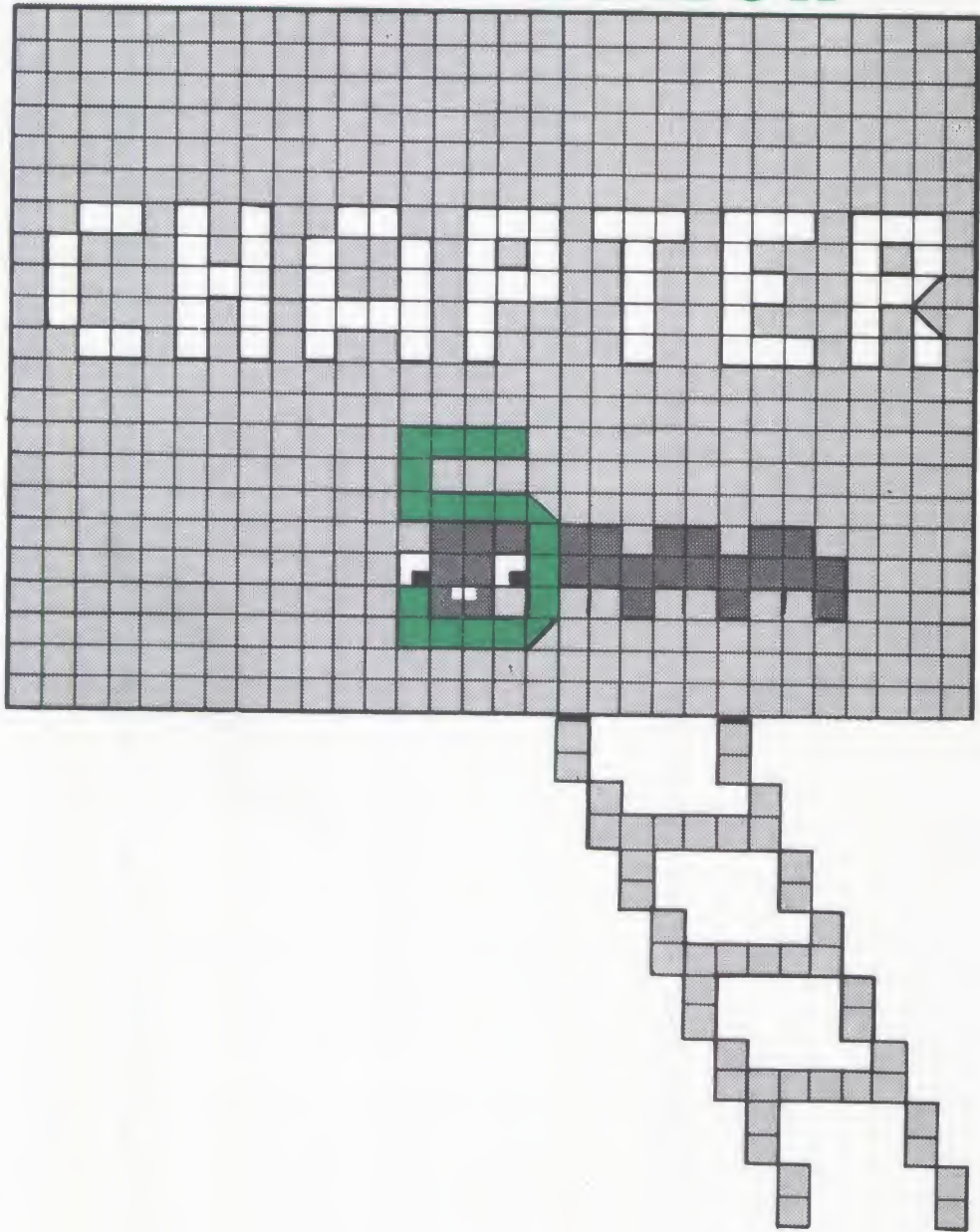
You may wish to familiarize yourself with the other graphics characters used for drawing. Remember to hold down the CTRL key as you press a character key. Many of the games you'll type in from magazines use these graphics characters.

Press the CTRL and the CAPS LOWR keys at the same time to "lock" the keyboard. You can now print the control characters without having to hold down the CTRL key the whole time. To return to upper case letters, press the SHIFT/CAPS LOWR keys, or press the SYSTEM RESET key.

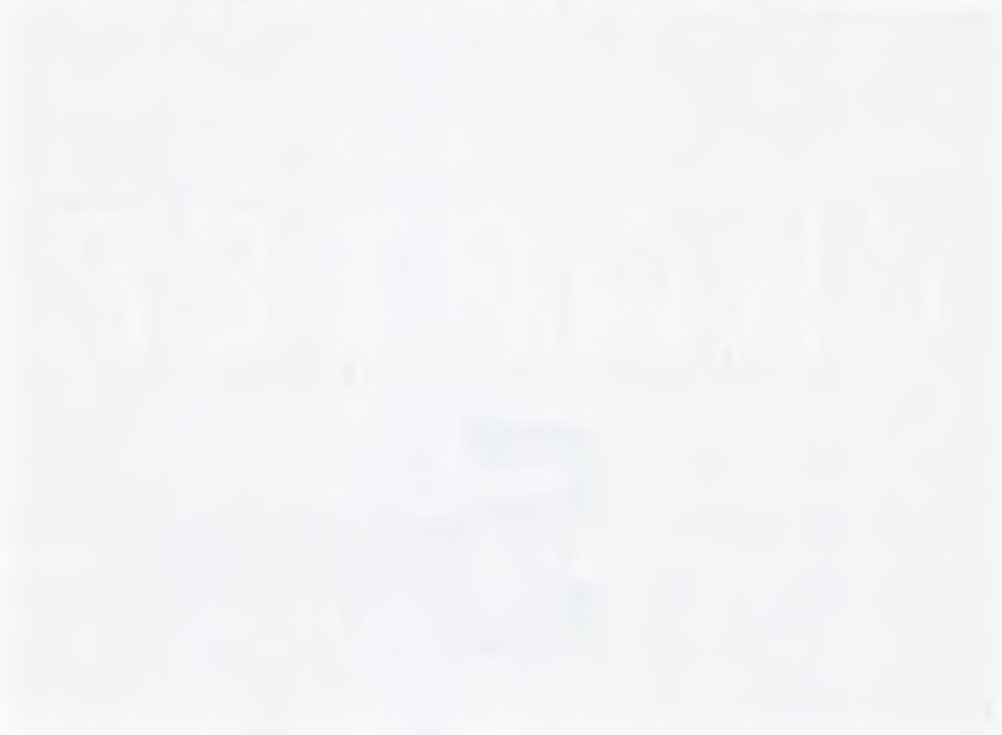
Before we leave this chapter, did you notice in the last program that you had the cursor in the lower left-hand corner? Do you recall how to remove it? Use POKE 752,1. It will be easy to put it into your program because the line numbers are on multiples of ten. Programmers do this so that later on, they can add lines. Use multiples of ten, also, in your own programming. If you read computer books and magazines, you'll see this is generally how it is done.

Now, on to the subject of the text window.

THE TEXT WINDOW



THE LEFT BRIDGE



The Text Window

Before you advance to the other graphics modes, you should learn about the text window. You didn't need to know about it in GR.0 because GR.0 doesn't have a text window. Many of the other modes do have one, though, so you should learn what it is and how it's used.

Since it's more effective for me to explain and demonstrate at the same time, please type in the following program. (If I ask you to type in a program and you already have one in the computer, you should use NEW to erase the previous program from your computer.)

```
10 GRAPHICS 0
20 PRINT "TECHNOLOGY"
30 GOTO 30
RUN
```

Remember, if you don't "position" your message (tell the computer the proper column and row coordinates) the word will be printed in the upper left-hand corner of your GR.0 screen.

Press BREAK or SYSTEM RESET because the program is still controlled by line 30. LIST it. Move your cursor and change the program to GR.1. RUN it.

Now the word is printed in the text window which is four lines of GR.0 at the bottom of your GR.1 screen. Print here whatever you would print on a regular GR.0 screen. It is important to know about this text window (hereafter called the TW) because most of the drawing modes won't let you write on the drawing screen, yet you may wish to communicate with the user (the person using your program). You'll see several examples of the usefulness of this message space as you continue through the book.

5 THE TEXT WINDOW

Press BREAK and LIST the program. Your computer reverts (defaults) to a GR.0 screen when you press the BREAK key. Since you already have four lines of GR.0 at the bottom of the screen, your computer will LIST the program there. This will be handy at times, but for now, let's change the screen to a complete GR.0 screen. Do you remember how to do this?

Press the SYSTEM RESET key, then LIST the program. This is the method you probably will use most often to clear the screen and LIST your program, unless you have +16 following the GRAPHICS number. In that case, use the BREAK key.

Change your program to GR.3, 4, 5, 6, and 7, and RUN each one. Through GR.7, you have a TW. GR.8 also has a TW, but it works differently so we'll study it separately. Readers who have the XL machines also have a TW in GR.14 and 15. Change the modes to GR.14 and GR.15 and RUN them.

LIST the program, and make the following changes:

```
10 GRAPHICS 1
20 PRINT #6;"TECHNOLOGY"
RUN
```

When you use PRINT#6; you address the "drawing" portion of your screen. You did not put in a POSITION statement, so the word is placed in the upper left-hand corner of the screen. It is also in color. We'll explain why it's in color in the next chapter (on GR.1 and GR.2). Change the program to GR.2 and RUN it. The word is still there, but each letter is much larger (we already learned how it is that letters can be different sizes).

REVIEW

To summarize what you have seen so far:

- 1) GR.0, GR.1, and GR.2 are the text modes. You write on these screens.
- 2) GR.1 through GR.8, GR.14 and 15 have what is called a text window (TW) for writing messages on the lower part of the screen.
- 3) You use PRINT "(words)" to display messages in GR.0 and in the TW in GR.1 through GR.8, GR.14 and 15.
- 4) You use PRINT#6; "(words)" to put messages on the main part of the screen in GR.1 and GR.2. These messages will automatically be in color. We'll discuss this thoroughly in the following chapter on GR.1 and GR.2.

You can also put messages on the screen and in the TW simultaneously, if you wish. Type in this program, please:

```
10 GRAPHICS 1
20 ? #6;"THIS IS THE SCREEN"
30 PRINT "THIS IS THE TEXT WINDOW"
40 GOTO 40
RUN
```




You now see both sentences; one is on the screen, the other is on the TW.

Try changing line 10 to GR.2 and RUN it. The result is the same except the letters on the screen portion are larger.

There is one more thing you can do with the TW: you can change its color and the color of the words printed there.

In GR.0, you recall, the background is controlled by SE.2,COLOR,BRIGHTNESS. Since there are sixteen colors (0 through 15) and eight brightnesses (0 through 14, even numbers only) you have 128 possible colors for the background. And since the TW is actually four lines of GR.0, you have that same number of colors available in the TW. You can change the color of it with SE.2,COLOR,BRIGHTNESS.

Use NEW to clear your computer and type in the following program, please:

```
10 GRAPHICS 1
40 PRINT "GR.1"
50 GOTO 50
RUN
```

You have the regular GR.1 screen with a message in the TW. Now add this to the program, please:

```
20 SETCOLOR 2,0,2
RUN
```

You see the same message, but the background color of the TW has changed. If you desire, you can change the color and the brightness and RUN the TW through the 128 colors, but that would be tiresome and repetitious since you've already worked on the background color in GR.0. However, try two or three

5 THE TEXT WINDOW

changes just to keep your hand at it. Remember that the combination of a color and a brightness determines the color you'll see. Take a minute off, and try a couple of changes.

Now let's change the brightness of the *message* you print. Recall that in GR.0 this is done with SE.1,0,BRIGHTNESS. Since the TW is four lines of GR.0, use the same formula. Add this, please:

```
30 SETCOLOR 1,0,0
RUN
```

The user controls the brightness/luminance/intensity of the message printed in the TW. If you come across some early programs (a number that are in the public domain, for instance), you'll see that the programmers often used a "regular" (white on blue) GR.0 TW to interact with the user. I think it looks much better to dress up your TW with a background color and a foreground (message) color.

Thus far we've used GR.1 only for the demonstration, but you can use color exactly the same way in GR.1 through GR.7, GR.14 and 15. Make the following changes to your program, please:

```
10 GRAPHICS 5
40 ? "GR.5"
RUN
```

The background color and the intensity of the GR.5 statement remain the same as they were in the previous program; this will be the same for all graphics modes 1 through 7, 14 and 15. Experiment with more of the modes, when you can.

The knowledge at your command gives you flexibility in your programming. You can, for instance, draw a picture in GR.7 in a certain color with a different color for the background. Then you can use the opposite approach in the TW: use the drawing color for the background in the TW, and the original background color for your messages. Or, you can use the same background and foreground colors in the TW that you use on the drawing screen. Or, you can pick different colors that harmonize or contrast. The TW coloring offers many possibilities. When you've learned to draw, you can work on color in the TW. You'll see color used in the TW when you do the "Multiplication Slave" program later on in the book.

There are two additional points to keep in mind. The first is that using SE.1,0,4 causes the message to be invisible; the second is that using the same brightness for SE.1 and SE.2 also causes the message to be invisible.

REVIEW

You've learned that a TW is four lines of GR.0

You've learned to print messages on it.

You've learned to control the color of the background of the TW.

You've learned to control the color of the foreground (messages).

You've learned to control the TW in GR.1 through GR.7, 14 and 15.

What about GR.8 through GR.11? GR.9, 10 and 11 do not have a TW. GRAPHICS 8 does, but as previously mentioned, color works a bit differently. Therefore, let's devote a little time to color in the TW in GR.8.

The GRAPHICS 8 Text Window

In GR.1 through GR.7, GR.14 and 15, you have a great deal of control over the TW. You can make its color different from the rest of the screen and you can change the brightness of the words you display there from that of the words or pictures displayed on the screen. In GR.8, you can do neither.

The color of the TW will be the same as the color of the background on the main portion of the screen. If, for example, you choose a reddish background color, the TW will also be reddish.

The color and the brightness of the words in the TW will be the same as the color and brightness of the lines you draw on the main portion of the screen. In effect, SETCOLOR 1,COLOR,BRIGHTNESS controls the drawing color *and* the TW word color.

Type in this program for a demonstration, please:

```
10 GRAPHICS 8:COLOR 1
20 SETCOLOR 1,14,0
40 PLOT 160,0:DRAWTO 160,159
```

Notice the colon (:). You use it to separate two different statements. Your computer has rigorous demands; one is that it does just one thing at a time. If you want it to do something new, you must type a new line number or separate the two statements with a colon. The PLOT statement tells the computer to place a dot of color at column 160, row 0. That's one entire task. The DRAWTO statement tells the computer to DRAW a line from the earlier dot of color to column 160, row 159. You must separate the two statements with a colon.

```
50 PLOT 161,0:DRAWTO 161,159
60 PRINT "THE UNINSPIRING TEXT WINDOW"
70 PRINT "IN GRAPHICS EIGHT."
```

When you place words in two separate lines, your computer will print them on two separate lines, one after the other.

```
80 GOTO 80
RUN
```

The color of the line drawn down the center of your screen is the same as the color of the words in the TW. Make the following change, please:

```
20 SETCOLOR 1,14,14
RUN
```


5 THE TEXT WINDOW

This changed the brightness of the line drawn *and* the brightness of the words in the TW. You cannot change the color of the words without changing the color of the lines that you draw; the two work together. Also, you cannot change the brightness of one without changing the brightness of the other. Make this change, please:

```
20 SETCOLOR 1,3,8  
RUN
```

You changed the color and brightness of the line, plus the color and brightness of the words in the TW.

To repeat—the color and the brightness of your drawing will always be the same as the color and brightness of the words in the TW.

The background works in the same way. Whatever color and brightness you choose for the background of the drawing portion of your screen will be the same color and brightness for the background of your TW. Add the following, please:

```
30 SETCOLOR 2,8,14  
RUN
```

You changed the background of the screen *and* the TW. (Notice that adding a background color affects the foreground color.) There isn't much room for creativity in the use of the TW in GRAPHICS 8, but at least you *have* a TW.

One cautionary note before we leave this discussion of the text window. If you change the color of the TW or the intensity of the letters you print there, you *may* change the colors of the major portion of the screen. This will depend upon the graphics mode and the color registers you use.

GRAPHICS ONE AND TWO





Graphics One and Two

We'll study these two graphics modes together because:

- 1) they are both text modes (you put words on the screen);
- 2) you can use special graphics characters for drawing; and
- 3) color works in the same way in these two modes.

Let's take a look at the size of the grids in these modes:

GR.1 has 20 columns and 20 rows.

GR.1+16 has 20 columns and 24 rows. Remove the four GR.0 lines at the bottom of the screen by putting in +16, giving you four more lines of GR.1.

GR.2 has 20 columns and 10 rows.

GR.2+16 has 20 columns and 12 rows. In this case, you have removed the four GR.0 lines, but four GR.0 lines are equivalent to only two GR.2 lines because of the size of the graphics grid. You learned about this in Chapter 2 ("Graphics Modes"). So, you go from 10 rows (GR.2) to 12 rows (GR.2+16).

Type in the following program and you'll see the difference between printing in the major portion of the screen and the TW.

```
10 GRAPHICS 1
20 PRINT #6;"SCREEN"
30 PRINT "TEXT WINDOW"
40 GOTO 40
RUN
```

There are two graphics modes on your TV or monitor; this is called a split

6 GRAPHICS ONE AND TWO

screen. You have the GR.1 screen and the GR.0 TW. List your program, change GR.1 to GR.2, then RUN it.

A difference exists in the size of the letters that are on the screen. (We discussed the reason for this earlier.) For a quick review, the more boxes there are on a screen grid, the smaller the characters will be. When there are fewer boxes, the characters will be larger. Since GR.2, for example, has half the boxes of GR.0, the letters will be twice the size in GR.2.

Another way of looking at it is that GR.1 characters are the same height but twice as wide as GR.0 characters. In GR.2, the characters are twice as high and twice as wide as they are in GR.0. Since color works the same in these two modes, let's take a look at it.

First Method: Using PRINT #6; and Upper Case Letters

In GR.0, you use PRINT "XXXX". In GR.1 and GR.2, you use PRINT #6; "XXXX". Type in the following program so you can learn to use this method:

```
10 GRAPHICS 1+16 (The +16 removes the TW at the bottom.)
20 POSITION 6,5
```

The invisible cursor moves over to column 6 and then down to row 5. Although you typed in column 6 and row 5, they are actually the seventh column and sixth row because numbering begins with 0. In the future, I'll use the POSITION numbers for the column and row numbers when I refer to them.

```
30 PRINT #6;"HELLO"
```

Use the command PRINT #6; to put characters on the screen. When you follow this command with capital letters, you use color register 0, which is automatically set to display orange. Thus you have the word **HELLO** in orange.

```
40 POSITION 8,10
```

This puts the invisible cursor in column 8 and moves it down to row 10.

```
50 PRINT #6;"(press the CAPS LOWR key and type in the follow-
ing word) how" (now press the SHIFT key and the CAPS LOWR
key simultaneously so your computer will return to upper case)
```

While you are entering this in the computer, you'll have the word "how" in lower case letters surrounded by quotation marks. When you press the CAPS LOWR key, you choose color register 1, which is green, so you will have a green **how**.

```
60 POSITION 11,13
```

Puts the cursor in column 11 and moves it down to row 13.


```
70 PRINT #6;"(now use the DOM key, then type in the following
word) ARE" (then press the DOM key again.)
```

You should have the word "are" in quotation marks. When you follow PRINT #6;" with the DOM key, you will use color register 2—which is blue—so you should have a blue are.

```
80 POSITION 14,16
```

Move the cursor to column 14, row 16.

```
90 PRINT #6;"(now press the DOM key once, release it and press
the CAPS LOWR key, then type in the following word) you"
(then press and hold down the SHIFT key as you press the CAPS
LOWR key. Press the DOM key.)
```

This returns your computer to normal from inverse video and from lower to upper case. When you use the DOM key and the CAPS LOWR key, you select color register 3 (which is red) and you should have a red you.

```
100 GOTO 100
RUN
```

Onscreen, you should see an orange **HELLO**, a green **HOW**, a blue **ARE**, and a red **YOU**. All words should be in upper case letters.

Don't get confused. Even though you used the CAPS LOWR key, your computer printed in capital letters. The CAPS LOWR key chooses a color register, *not* lower case letters, on the drawing screen in GR.1 and GR.2. To place actual lower case letters there, you have to do something different, which you'll learn to do shortly.

GR.1 and GR.2 work exactly the same way. LIST your program. Use the CTRL in combination with the arrow keys to move your cursor up to line 10. Change it to GR.2+16; press RETURN.

As you recall, in GR.2+16 you have only twenty columns and twelve rows. This means you'll have to change the POSITION statements so the message will fit on this smaller screen.

Move your cursor to the POSITION statements and change them, please:

```
20 POSITION 8,2
40 POSITION 8,4
60 POSITION 8,6
80 POSITION 8,8
RUN
```

Since there are only twelve rows in GR.2+16, we had to make the above changes; otherwise we would have received another "ERROR" message because the words would not have fit on the screen.

6 GRAPHICS ONE AND TWO

You wind up with the same colors for the words, but the words are twice as high as they were in GR.1.

When you design a nifty program, you should know how to set up a title page. Use NEW to erase old program lines from your Atari, then type in this program—an example of a title page:

```
10 GR.1
20 POS.5,5
30 ?"(DOM) WEREWOLVES (DOM)"
40 FOR WAIT=1 TO 500:NEXT WAIT
```

This is one way to use a FOR...NEXT loop and it's the easiest and least confusing. It's a simple command you give the computer that tells it to count from 1 to 500 and then go to the next statement. Use any numbers in here to make the wait longer or shorter.

```
50 POS.9,9
60 ?" (DOM and CAPS LOWR) by (DOM. SHIFT and CAPS
LOWR at the same time) "
70 FOR WAIT=1 TO 500:NEXT WAIT
80 POS.4,13
90 ?" (press CAPS LOWR key) wotza dimple (SHIFT and
CAPS LOWR simultaneously) "
100 FOR WAIT=1 TO 500:NEXT WAIT
RUN
```

You should have typed the words “werewolves,” “by,” and “wotza dimple” within quotation marks.

Wasn't that sneaky of me? This is not what we wanted. We wanted to make a title page in color—but look at what we have!

When you have a problem in a program, it is called a bug. The process of removing the bug is called debugging. What you must do is find out what's wrong with the program and correct it. The fault usually is *not* your computer's. You may, at times, be tempted to open the top of your computer and spray some insecticide in it, but most programming problems originate with the programmers or typists and not the machines.

LIST your program. Think about the problems. Take a few minutes and try to solve them by yourself.

Okay. Look at the “hello how are you” program. Notice what we've done there. Compare it to the program LISTed on the screen. Get the old noodle working. Make corrections. There's nothing this program encompasses that you haven't already learned. Good hunting!

Now that you're back, did you find all or most of the problems? If not, don't worry about it.

Look at line 10. There's something wrong with it. You need to add +16 to it to remove the TW. Do so and RUN it. Still not right?

LIST it again. Look at line 20. Seems okay, doesn't it? Go to line 30. There's something wrong here. Know what it is? Think about it.

Place your cursor in line 30 on top of the first set of quotation marks. Now press the CTRL key and hold it down as you press the INSERT key four times. Now type in #6. Press RETURN. RUN it.

Still wrong, isn't it? It's wrong because we didn't change the other PRINT statements to PRINT #6;. Do so and RUN it again.

Still a problem? You don't want to return to a GR.0 screen after your title page, so you'll have to do something to keep the GR.1 screen. Do you recall what to do?

Right! Type in another line.

```
150 GOTO 150
RUN
```

Works perfectly, doesn't it? Fine. Now we want to make the words fatter. Let's change the program from GR.1+16 to GR.2+16. Make the changes, then RUN it.

```
ERROR- 141 AT LINE 90 staring you in the face?
```

Phooey! LIST it again; check line 90. It's okay now, isn't it? The problem isn't in line 90, it's in line 80. Remember we have only ten rows instead of twenty. What you'll have to do is change all the POSITION numbers. Work on it until you get the words positioned just where you want them. Here's a correct LISTing for the program:

```
10 GRAPHICS 2+16
20 POSITION 5,1
30 ? #6;"WEREWOLVES"
40 FOR WAIT=1 TO 500:NEXT WAIT
50 POSITION 9,5
60 ? #6;"by"
70 FOR WAIT=1 TO 500:NEXT WAIT
80 POSITION 4,9
90 ? #6;"wotza dimple"
100 FOR WAIT=1 TO 500:NEXT WAIT
150 GOTO 150
```

REVIEW

You now understand one way to work with color in GR.1 and GR.2, using:

- 1) PRINT #6; and capital letters to get orange letters (color register 0)
- 2) PRINT #6; and CAPS LOWR to get green letters (color register 1)

6 GRAPHICS ONE AND TWO

- 3) PRINT #6; and the DOM key to get blue letters (color register 2)
- 4) PRINT #6; and the DOM key with the CAPS LOWR key to get red letters (color register 3).

You've noticed that all characters are in upper case. Points 2) and 3) select colors, *not* lower case letters. How do you select lower case letters? The answer to that question is on its way.

Second Method: Using PRINT #6; and Lower Case Letters

Use POKE 756,226. A POKE addresses a certain location in your computer that issues instructions. In this case, the POKE tells the computer to select the lower case letters. There are some complications, however. We'll discuss them after you type in the following program demonstrating the use of color with lower case letters.

```
10 GR.1+16
20 POKE 756,226
```

This POKE tells the computer to print in lower case letters.

```
30 POS.6,3
40 PRINT #6;"LEARNING"
50 POS.8,7
60 PRINT #6;" (press CAPS LOWR key) about (press SHIFT
and CAPS LOWR keys simultaneously) "
70 POS.6,11
80 PRINT #6;" (press DOM key and release it) MY ATARI
(press DOM again) "
90 POS.7,15
100 PRINT #6;" (press DOM and CAPS LOWR keys) is fun
(press DOM, then SHIFT and CAPS LOWR at the same time) "
110 GOTO 110
Keeps screen from reverting to GR.0.
RUN
```

First, notice that whether we use upper or lower case letters in the program, we still end up with lower case letters when we run the program. That's due to the POKE, and because capital letters, CAPS LOWR, and DOM choose the colors the same way they do when we use them in the other programs. Our POKE in line 20 determined that lower case letters would be displayed on the screen.

What to do now with all the orange hearts? If you want to leave them in, that's fine. But if you don't, there are at least two ways to get rid of them.

One way is rather difficult. You transfer the character set from ROM to RAM, change the heart character to a space character, and then use the lower case letters. As I said, it's difficult and we won't run through it here.

Luckily, there's an easier way to get rid of the hearts. As with most shortcuts, however, you must pay a price. Type in this additional line:

```
25 SETCOLOR 0,0,0
RUN
```

The hearts are gone. Great! But missing also is the orange word "learning." Why? Because the color orange comes from register 0, and we have changed register 0 with the SETCOLOR 0 statement. Further, our SETCOLOR statement used 0,0 which makes *black* the background color. Now there is no orange color, so you lost the word "learning." You will have only three colors for your messages (green, blue, and red) when you use lower case letters. If three colors are enough for you when using lower case letters, that's the perfect solution. Later in this chapter, we'll talk about changing the three colors if you don't care for green, blue, and red.

There is another disadvantage to using the POKE 756,226 to print on the screen with lower case letters: you won't be able to print numbers at the same time you print lower case letters. So, you lose one color and you can't use numbers. To return to upper case letters inside a program, you must POKE 756,224. Add these lines, please:

```
110 FOR WAIT=1 TO 1000:NEXT WAIT
120 POKE 756,224
130 GOTO 130
RUN
```

First you see the message in lower case letters, there's a pause, then you see the message in upper case letters. To get the complete message back on the screen, you can move your cursor to the word "learning" in line 40, press the DOM and CAPS LOWR keys, then type "learning" again. Press the DOM, then the SHIFT and CAPS LOWR, and put in the final quotation marks. RUN it. It works!

REVIEW

So far you have been using:

- 1) Upper case letters and PRINT #6; (color register 0) to get orange.
- 2) CAPS LOWR letters and PRINT #6; (color register 1) to get green.
- 3) Upper case letters and the DOM key (color register 2) to get blue.
- 4) CAPS LOWR letters and the DOM key (color register 3) to get red.
- 5) The above four with POKE 756,226 to get real lower case letters in color.

Using Color in GR.1 and GR.2: First Method

Now we'll change the four colors (orange, green, blue, and red) by using the SETCOLOR statement. Recall that you select a color register with the SET-COLOR statement.

6 GRAPHICS ONE AND TWO



Please type in the following:

```
10 GRAPHICS 1+16
30 POSITION 5,4
40 ? #6;" (CAPITAL LETTERS) LEARNING"
60 POSITION 5,8
70 ? #6;" (CAPS LOWR) color (SHIFT and CAPS LOWR) "
90 POSITION 5,12
100 ? #6;" (DOM) IS (DOM)"
120 POSITION 5,16
130 ? #6;" (DOM and CAPS LOWR) challenging (DOM,
SHIFT, and CAPS LOWR)"
140 GOTO 140
RUN
```

You see the regular orange, green, blue, and red words. Add the following, and you'll change those colors:

```
20 SETCOLOR 0,12,2
50 SETCOLOR 1,13,14
80 SETCOLOR 2,6,4
110 SETCOLOR 3,7,8
RUN
```

On my TV, the colors have become a dark green, a light yellow, a purple, and a pretty shade of blue.

You can also change the color of the border/background. Change line 10 to the following:

```
10 GRAPHICS 1+16:SETCOLOR 4,15,14
RUN
```

Color register 4(SETCOLOR 4) changes the border/background color. Changing the background color will affect the foreground color, as you plainly see. If you plan to use a background color other than black, put on the background color *first*, then add the other colors one at a time, **RUN**ning the program with each addition to get the colors you like.

Remember that you use this SETCOLOR technique in both GRAPHICS 1 and 2. GRAPHICS 1 and 2 are twins; one's just a bit fatter than the other.

With this knowledge of color registers, SETCOLOR statements, DOM, CAPS LOWR, etc., you are now the master at putting messages on your GR.1 and 2 screens to:

- 1) welcome visitors for dinner;
- 2) wish happy birthday to your girlfriend, boyfriend, husband, wife, friend, etc.;
- 3) make snazzy title pages; or
- 4) write poetry.

Be creative. Experiment and find new ways to use these skills.

REVIEW

Upper case letters with the PRINT #6; statement use color register 0, which gives us an orange color. To change this color, use the word SETCOLOR with the same register number. In this case, we would say SETCOLOR 0,COLOR, BRIGHTNESS. *Remember*, the colors are numbered from 0 to 15; the brightnesses are from 0 to 14, even numbers only.

Lower case letters with the PRINT #6; statement use color register 1, which gives us a green color. Change this color by using the register number in a SETCOLOR statement: SETCOLOR 1,COLOR,BRIGHTNESS. In the last program, you used SETCOLOR 1,13,14, which changed the green to a yellow.

Upper case letters and the DOM key with the PRINT #6; statement give us blue letters. This is color register 2. Change the color by using SETCOLOR 2,COLOR,BRIGHTNESS.

Lower case letters and the DOM key with the PRINT #6; statement give us red letters. This is color register 3, so use SETCOLOR 3,COLOR,BRIGHTNESS to change the color. When you use SETCOLOR with the appropriate register number, you can change the color in the paint jar.

6 GRAPHICS ONE AND TWO

Using Color in GR.1 and 2: Second Method

The second way to use color in GR.1 and 2 is a bit more complicated. It will be easiest if you type in the following program, then we'll discuss what we've done:

```
10 GRAPHICS 2+16:POKE 756,226
```

The POKE means we'll use lower case letters.

```
20 SETCOLOR 0,0,0
```

Removes the hearts. Also uses one of our color registers.

```
30 SETCOLOR 1,12,12
```

A Pretty lime green.

```
40 COLOR 81-64
```

```
50 PLOT 9,4
```

Places a dot in the middle of screen.

```
60 COLOR 87-64
```

```
70 PLOT 10,4
```

```
80 COLOR 69-64
```

```
90 PLOT 11,4
```

```
100 GOTO 100
```

```
RUN
```

Recall the window we drew in GR.0? This program paints one-third of the same window in lime green. The first SETCOLOR statement (line 20) eliminates the hearts (since we're in lower case), and the second one (line 30) colors the window. The color statements that follow choose the symbol to be printed and the color register. SETCOLOR 1,12,12 in line 30 changed the color in that register. You'll now need pages 55 and 56 in your *Atari BASIC Reference Manual*. Please turn to them since they will facilitate your understanding of what we're about to discuss.

Look at page 55 and observe that each character has a number. The number for a plus sign (+) is 11. The number for an upper case "L" is 44; for a lower case "t" it is 116, etc.

Notice also that these characters are displayed in columns. They're divided this way for a specific reason, not just for the sake of display. These four columns correspond to the four conversions on page 56.

The first two columns on page 55 are upper case; the third and fourth are lower. To use the third and fourth columns, you must use lower case. In our program, we POKEd 756,226 (line 10) so we could use lower case. We chose the characters from column three, then we turned to page 56. It is divided into four conversions that correspond to the columns on page 55. In other words, column one on page 55 works with conversion one on page 56, column 2 on page 55 works with conversion two on page 56, etc.

In GRAPHICS 0, you drew a picture of a house with a window in it. You used the CTRL key combined with letters to draw the picture of the window. The color of the window depended upon the background color. In GR.1 and 2, you can still use these character graphics to draw pictures, but you now have four colors for your drawing. There is one background/border color and three colors for your characters if you use a register to remove the hearts. Drawing a picture in GR.0 is simple, but with only one color to work with, the results are less than smashing.

In GR.1 and 2, you still use the same characters for the window but, as you might imagine, we have to work harder since the results will be much finer and so much more colorful than they were in GR.0—especially if you use this technique.

Look at column 3 on page 55 of the *Manual*. Here we see the graphics characters we want for drawing our picture. We know the CTRL/Q will give us the upper left-hand corner of the window. The number next to the Q is 81.

Now turn to page 56. Since we used column 3 on page 55, we must use conversion 3 on page 56. Look all the way to the left and you'll see the MODES divided. First, there is MODE 0, below which are MODES 1 and 2. We are in GR.2 in our program, so we must use the lower half of the character/color assignment chart. Notice that MODES 1 and 2 are divided into four rows. Each row is a SETCOLOR row:

SETCOLOR 0 = orange

SETCOLOR 1 = green

SETCOLOR 2 = blue

SETCOLOR 3 = red

Since we used column 3 for our character (the upper left-hand corner of the window), we must use conversion 3 to get the proper number for the character. We have four choices:

#-32

#-64

#+96

#+64

(If you have an earlier *Atari BASIC Reference Manual*, the numbers in it are incorrect. The above four numbers are the correct ones.)

The number we choose will be subtracted from or added to the number for the "Q," which is 81. Which minus or plus do we use? Whichever color we want. One note of caution, though. Look at line 20 of our program. We have already used SETCOLOR 0 to remove the hearts that automatically fill the screen when we go to lower case letters. Therefore, we can't use SETCOLOR 0.

Look at line 40 in the program. The statement is COLOR 81-64. You know that 81 is the number for the graphics character for the CTRL/Q, which is the upper left-hand corner of the window. We told the computer to subtract 64 from the number. Look at conversion 3 on page 56. The statement there is #-64. This

6 GRAPHICS ONE AND TWO

means take the number from page 55 and subtract 64 from it. Go left across the ROW from #-64 and you'll see SETCOLOR 1. SETCOLOR 1, you know, is green.

The next step is to use the statement COLOR followed by the number from page 55, which is increased or decreased by the number in conversion 3 on page 56. If, for example, you want to use the CTRL/Q character but you want it in blue, what would your statement be? Try to figure it out before you look at my answer.

Take the number 81 from column 3 on page 55, then look at conversion 3 on page 56. Look down that conversion until you are opposite SETCOLOR 3, which we know is blue. Go back across the row to conversion 3 and you see the message #+96. This means you will have to make the statement COLOR 81+96 to get blue. If you put this statement (COLOR 81+96) in line 40, and RUN the program, you'll see the upper left-hand corner of the window in blue.

So you can draw in GR.0 with the graphics characters in one color, or you can draw with the same graphics characters in GR.1 or GR.2 in three colors plus the background color which, as you know, affects the foreground colors.

The program that follows shows you how to use the "COLOR + or - technique" to print letters and characters. You'll see a picture of a window surrounded by the word **window**. Then we'll use this program as the basis for a new technique in changing color.

The length of the program may look formidable, but there is so much repetition that you can use our trick and easily type in the program by doing the following.

Please type in lines 10 through 60. Look at line 50 and line 70. They are practically the same. The two differences are the number immediately following the word COLOR, and the letter following the abbreviation REM. (We'll explain the REMs later when we discuss the program.)

All you have to do is place the cursor on the 5 in line 50 and change it to a 7. Now use the CTRL/right arrow keys and move the cursor to the second digit after the word COLOR. Compare lines 50 and 70. One has 81-64:REM 0, and the other has 87-64:REM W.

All you have to do is change one digit and the one letter.

Type a 7 on top of the 1. Use the CTRL/right arrow keys to place the cursor on top of the "Q." Type in a "W." Press RETURN.

Your program contains lines 50 *and* 70 and the cursor is now on the number 6. Compare lines 60 and 80. Again, they are practically the same. Only one digit is different.

Type an 8 over the 6. Use the CTRL/right arrow keys to place the cursor on the number following the word "PLOT." Type a 9 over that 8. Press RETURN and LIST your program.

Check to make sure that the lines on the screen match the lines in the book. If they don't, use the editing features to make them match.

Place the cursor on the 7 in line 70. Type in a 9.

Compare lines 70 and 90.

Make the necessary changes.

Press RETURN.

Compare lines 80 and 100.

Just continue the same process through line 220.

Then, type in lines 230 and 240.

Lines 250 through 640 can also be easily typed in using the trick technique. Type in line 800, and you're finished typing the program.

This trick will come in handy throughout the workbook, which contains a number of long programs.

Type in the program and make sure you have an hour or so to spare. You'll learn quite a few things from this program and it would be best to absorb all of them at one sitting, if possible. If you can't afford that much time right now, an alternative is to type in the program and SAVE it so you can come back to it later.

The following program will draw a window with the word **window** framing it. (Remember that the abbreviation PL. stands for PLOT.)



```
10 GRAPHICS 2+16:POKE 756,226
20 SETCOLOR 0,0,0
30 SETCOLOR 1,2,10
40 REM PLOTS FOR WINDOW
50 COLOR 81-64:REM Q
60 PLOT 8,4
70 COLOR 87-64:REM W
80 PLOT 9,4
```

6 GRAPHICS ONE AND TWO

```
90 COLOR 69-64:REM E
100 PLOT 10,4
110 COLOR 65-64:REM A
120 PLOT 8,5
130 COLOR 83-64:REM S
140 PLOT 9,5
150 COLOR 68-64:REM S
160 PLOT 10,5
170 COLOR 90-64:REM Z
180 PLOT 8,6
190 COLOR 88-64:REM X
200 PLOT 9,6
210 COLOR 67-64:REM C
220 PLOT 10,6
230 REM PLOTS FOR WINDOW
240 REM MAKE WORD BLUE
250 COLOR 119+96:REM W
260 PLOT 2,0
270 COLOR 105+96:REM I
280 PLOT 5,0
290 COLOR 110+96:REM N
300 PLOT 8,0
310 COLOR 100+96:REM D
320 PLOT 11,0
330 COLOR 111+96:REM O
340 PLOT 14,0
350 COLOR 119+96:REM W
360 PLOT 17,0
370 COLOR 105+96:REM I
380 PLOT 2,2
390 COLOR 110+96:REM N
400 PLOT 2,4
410 COLOR 100+96:REM D
420 PLOT 2,6
430 COLOR 111+96:REM O
440 PLOT 2,8
450 COLOR 119+96:REM W
460 PLOT 2,10
470 COLOR 105+96:REM I
480 PLOT 5,10
490 COLOR 110+96:REM N
500 PLOT 8,10
510 COLOR 100+96:REM D
520 PLOT 11,10
530 COLOR 111+96:REM O
540 PLOT 14,10
550 COLOR 119+96:REM W
560 PLOT 17,10
570 COLOR 105+96:REM I
580 PLOT 17,2
590 COLOR 110+96:REM N
```

```

600 PLOT 17,4
610 COLOR 100+96:REM D
620 PLOT 17,6
630 COLOR 111+96:REM O
640 PLOT 17,8
800 GOTO 800
RUN

```

One thing I should mention is the large number of REM statements (which are ignored by the computer when it RUNs a program). The REMarks are for the benefit of the programmer, and are used for a variety of reasons. I'll discuss the three most important ones here.

In this case, the REMs tell you which number to choose to get the graphics symbol or the letter on the screen. If, for example, you type in the program but find that the middle of the window is out of place, you could LIST your program. Remembering that "S" is the middle of the window, you then could go to the REMark statement showing where the "S" is and debug your program by changing the X (column) and Y (row) coordinates. If you did not have a REMark at line 130, you'd have to go through the program line-by-line, remembering that you first used a "Q," and then a "W," etc. The REMarks, therefore, aid in the debugging process.

Here's another reason for putting in the REMs. Let's assume that you SAVE this program, forget about it for six months, then decide to change the shape of the window (you want to make it longer or wider). You could easily do this because your REMs remind you where the instructions for the window start.

Another time to use REMs is to remind yourself that a part of the program deals with a certain formula. If you haven't made a REMark, you'll have to go through the program line-by-line to find where the formula begins. These are three very good reasons to use REMarks. Now, on to the program.

We used the letters Q,W,E,A,S,D,Z,X, and C to draw the window. Notice that the word **window** is in lower case. Do you recall the reason? Right. When you use the graphic character symbols you must use lower case, and when you do you get lower case letters as well.

First, you have decided to use graphics characters to draw a window. You know which characters you want to use, so go to column 3 on page 55 and look up the number for the character. Then turn to page 56, conversion 3, and choose the SETCOLOR. Match it with the *number +* or *number -* in conversion 3. Then type in COLOR 84+96, or whatever. Finally, PLOT where you want that character displayed.

That's exactly what was done in this program. As an example, look at line 250. We wanted to display the letter "w," so we chose COLOR 119+96 to make it blue. In line 260, we placed the blue "w" at column 2, row 0. And that's that.

If you're still confused about how to do this yourself, here's an easy way to see what is going on. This should enable you to write a similar program.

6 GRAPHICS ONE AND TWO

Type in the following at the end of your program, please:

```
65 FOR WAIT=1 TO 300:NEXT WAIT
85 FOR WAIT=1 TO 300:NEXT WAIT
105 FOR WAIT=1 TO 300:NEXT WAIT
RUN
```

These three wait loops slowed down the program considerably so you can see what lines 50-60, 70-80, and 90-100 do. Now check those lines against pages 55 and 56 in the *Manual*, and you'll understand the process much better.

Keep this in mind: whenever you encounter part of a program that doesn't make sense, use one or more wait loops to slow down the execution. It may clarify things for you.

A further word on character graphics: if you wish to work with these characters, I suggest you use the CTRL/character approach. It'll be much less time consuming; all you have to do is press the CTRL key together with a character key to place a graphics character on the screen in GR.0, GR.1, or GR.2.

If you have sufficient time now, let's work with this program and learn some new things. If you're short on time, SAVE the program and come back to this page later.

We have used -64 for the picture of the window. If you refer to page 56 in your *Manual*, you'll see this (from conversion 3) chooses SETCOLOR 1, which we know is screen. But take a look at line 30. We changed the color green to a yellow or orange hue, depending on your TV. Let's review the SETCOLOR statement, then we'll make the window cycle through its sixteen colors.

REVIEW: SETCOLOR R,C,B,

R is the Register number. There are five registers: 0, 1, 2, 3 and 4. We are using SETCOLOR (Register) 1 for the window.

C chooses the Color we wish to use. The colors are from 0 to 15. Check the front of the book to review them, if you need to.

B is for the Brightness or luminance of the color which, as you recall, will change the color. The numbers in this slot range from 0 to 14, even numbers only. (If you type in an odd number here, the computer will pick the lowest even number. For example, if you put in the number 7, your computer will treat it as if it were a 6.)

This recapitulation should help you understand how we can make the window cycle through its sixteen colors. Add this line to the program, please:

```
650 FOR WAIT=1 TO 200:NEXT WAIT
```

When the computer has RUN through the program, it waits until it has counted to 200. Add the following, please:

```
660 FOR C=0 TO 15
```

This is a new way to use the FOR...NEXT loop. This statement tells the computer that the letter "C" has the value of 0, 1, 2, 3, etc., all the way up to 15. Add this line, please:

```
680 SETCOLOR 1,C,2
```

SETCOLOR 1 is the register we used for the window. C, we know, equals 0 to 15. Thus far, we are addressing the register that controls the color of the window. Also, we are saying that the color we want is not one number, but 16 (C=0 TO 15). The 2 controls the brightness; it is very dark. Type in the next line, please:

```
710 NEXT C
```

This is the other half of the FOR...NEXT loop that began in line 660. This instruction tells the computer, "Okay, you made the color number 0, now make it a 1." The computer will go back to line 660, make C equal 1, then will go through to line 680, making the color equal 1. When it gets to line 710, it is told to return to the beginning of the FOR...NEXT loop again and make the color one number higher until it has returned to line 660—making C equal to all numbers from 0 to 15. Then it will go on to line 800 and patiently sit there. The screen will not clear and return to a GR.0 screen. The picture will stay on the screen with SETCOLOR 1,15,2 for the final color of the window. It is now color 15 because of the FOR...NEXT loop.

RUN

First, the computer draws the picture of the window. Second, it puts the word **window** on the screen. Third, it waits until it has counted to 200. Fourth, it changes the color of the window to COLOR 1, then COLOR 2, then COLOR 3, etc., until it reaches COLOR 15. Then the picture stays on the screen.

The color of the window *did* change, didn't it? The color cycling is so swift, it might be a good idea to slow it down. How can we do that? Correct. Use another FOR...NEXT loop to make the computer count to itself.

We want the computer to make the window COLOR 0, then pause; make it COLOR 1, then pause; make it COLOR 2, then pause. Have it continue in this way until the cycling stops at COLOR 15. To make this work, we'll have to put in our counting loop *after* the computer has made one color change and *before* it makes the next one. Look at the end of the program and figure out where to put the counting loop. Take your time and think the problem through.

Now that you've thought about it and probably have figured out a solution, this is how I solved it:

```
690 FOR WAIT=1 TO 200:NEXT WAIT
RUN
```


6 GRAPHICS ONE AND TWO

If you put the wait loop *after* line 710, the computer will display all sixteen colors, then WAIT for a count of 200. That wouldn't help at all. (By the way, when you have one loop inside another, it's called a nested FOR...NEXT loop.)

So far, we've put the picture on the screen with the word in one color and the window in another. Next, we cycled the window through its sixteen colors, pausing after each color change. Now let's try something new and cycle the *word* through the sixteen colors.

We used SETCOLOR 1 for the window and +96 for the color of the word. If you look on page 56 you'll see that +96 uses SETCOLOR 2. to change the color of the word, we'll have to use SETCOLOR 2. Just change line 680 to:

```
SETCOLOR 2,C,2  
RUN
```

Now the window stays one color while the *word* cycles through all sixteen.

So, we have cycled the window, then the word, through the sixteen colors. Can you guess where we are headed? Yep—we will now cycle both the window *and* the word through the colors.

You'll have an easy time doing this. Type in the following, please:

```
670 SETCOLOR 1,C,2  
RUN
```

The word *and* the window cycled through the sixteen colors.

Let's make the BRIGHTNESS cycle also. (*Remember*, the brightnesses are 0, 2, 4, 6, 8, 10, 12, and 14.) Add this to your program:

```
665 FOR B=0 TO 14 STEP 2
```

The STEP 2 tells the computer to increment the change by two each time. In other words, the BRIGHTNESS will start at 0, go to 2, then to 4, etc. Make these changes, please:

```
670 SETCOLOR 1,C,B  
680 SETCOLOR 2,C,B  
700 NEXT B
```

Press SYSTEM RESET, and RUN your program.

Your computer will start the color at 0, BRIGHTNESS 0. Next it will display the other brightnesses (2, 4, 6, 8, 10, 12, and 14) then go to COLOR 1 and do the eight brightnesses, until it finally reaches COLOR 15 and BRIGHTNESS 14.

It takes a rather long time because it actually displays sixteen colors and the eight brightnesses for each color. That's a total of 16×8 , or 128 colors. Speed up the program by lowering the number in the FOR...NEXT loop in line 690.

You can also cycle through any number of colors or brightnesses by altering lines 660 and 665. Add the words STEP 2 to line 660; change the statement from STEP 2 to STEP 4 in line 665; RUN it. Your computer starts with COLOR 0, BRIGHTNESS 0, and will change to BRIGHTNESS 4, then BRIGHTNESS 8, and BRIGHTNESS 12. It will go to COLOR 2 (because of the STEP), then cycle through the brightnesses again: 0, 4, 8, 12, and so on. The last color number will be 14 because of our statement C=0 TO 15 STEP 2. The computer won't go beyond COLOR 14 because the next STEP would put it at 16, which is above the parameter we set in the C= statement. For the very same reason, it won't go beyond BRIGHTNESS 12.

In case you encountered a problem, these are the statements from line 650 onward for the program you just completed (including the alterations of lines 660 and 665):

```
650 FOR WAIT=1 TO 200:NEXT WAIT
660 FOR C=0 TO 15 STEP 2
665 FOR B=0 TO 14 STEP 4
670 SETCOLOR 1,C,B
680 SETCOLOR 2,C,B
690 FOR WAIT=1 TO 200:NEXT WAIT
700 NEXT B
710 NEXT C
800 GOTO 800
RUN
```

For those who know little about computers (and even for those who know quite a bit), this cycling of colors is pretty impressive. Yes, at times your Atari can be complicated to master, but it can do some neat stuff that other computers can't.

Before we leave GR.1 and 2, there are two more points to show you. If you wish, SAVE the preceding program, then type in NEW and press RETURN. Enter this program, please:

```
10 GRAPHICS 1+16:SETCOLOR 0,0,0
20 COLOR 35+64
30 PLOT 3,6
40 COLOR 47+160
50 PLOT 6,6
60 COLOR 44+192
70 PLOT 9,6
80 COLOR 47+160
90 PLOT 12,6
100 COLOR 50+64
110 PLOT 15,6
120 GOTO 120
RUN
```

On your Atari, you can use the letters of a word in a number of different colors. Try experimenting and come up with some interesting displays.

Now add this to line 10: POKE 756,226; RUN it again.

6 GRAPHICS ONE AND TWO

How did that happen? Baffled? You're in lower case so you should have used column 4 on page 55 of the *Manual*, and conversion 4 on page 56. You didn't, but you get the same letters—and you still get colors. I'll leave this problem for you to figure out. I'll give you a hint: look up the numbers in column 4 on page 55 for the word COLOR, then go to conversion 4 on page 56 (of the *Manual*) and figure out why you got the colors you did. Good hunting!



GRAPHICS THREE, FIVE, AND SEVEN



Graphics Three, Five, and Seven

Mode		Grid
GRAPHICS 3	has	40 columns and 20 rows.
GRAPHICS 3+16	has	40 columns and 24 rows.
GRAPHICS 5	has	80 columns and 40 rows.
GRAPHICS 5+16	has	80 columns and 48 rows.
GRAPHICS 7	has	160 columns and 80 rows.
GRAPHICS 7+16	has	160 columns and 96 rows.

The above modes are known as PLOTting modes. In these modes, you tell the computer to put a dot of a certain color at a particular point on the grid/screen; this is the way you draw pictures. Type in the following, please:

```

10 GRAPHICS 3
20 COLOR 1
30 PLOT 20,10
40 ? "COLOR 1 IS REGISTER 0:ORANGE."
50 ? "THE COLUMN IS 20, AND THE ROW IS 10."
60 GOTO 60
RUN

```

You should have an orange rectangle near the center of the screen, and text in the TW at the bottom. Run your cursor up to line 20. Change it to COLOR 2 and go to line 40, changing it to:

```

40 ? "COLOR 2 IS REGISTER 1:GREEN."
RUN

```

7 GRAPHICS THREE, FIVE, AND SEVEN

There should be a green rectangle. Now change line 20 to COLOR 3. Change line 40 to:

```
40 ? "COLOR 3 IS REGISTER 2:BLUE."  
RUN
```

Change line 20 to COLOR 0. Change line 40 to:

```
40 ?"COLOR 0 IS REGISTER 4:BLACK"  
RUN
```

Your last change picked COLOR 0 (register 4) which is black, the background color. You should see no rectangle. What follows is the complete last program:

```
10 GRAPHICS 3  
20 COLOR 0  
30 PLOT 20,10  
40 ? "COLOR 0 IS REGISTER 4:BLACK"  
50 ? "THE COLUMN IS 20, AND THE ROW IS 10."  
60 GOTO 60
```

From this little program, you can see that:

COLOR 1 selects register 0, which contains orange paint.

COLOR 2 selects register 1, which contains green paint.

COLOR 3 selects register 2, which contains blue paint, and
COLOR 0 selects register 4, which contains black paint (making the
rectangle invisible).

Please type in this program displaying the three colors. Press the SYSTEM RESET key, type NEW, then press RETURN. Be sure to use the special trick on this one, because the lines are practically the same.

```
10 GRAPHICS 3  
20 COLOR 1  
30 PLOT 15,5  
40 PLOT 25,5  
50 PLOT 15,15  
60 PLOT 25,15  
70 ? "THIS IS COLOR 1,REG.0:ORANGE."  
100 COLOR 2  
110 PLOT 15,5  
120 PLOT 25,5  
130 PLOT 15,15  
140 PLOT 25,15  
150 ? "THIS IS COLOR 2,REG.1:GREEN."  
180 COLOR 3  
190 PLOT 15,5  
200 PLOT 25,5  
210 PLOT 15,15  
220 PLOT 25,15
```



```
230 ? "THIS IS COLOR 3,REG.2:BLUE."
RUN
```

The program did exactly what you told it to: it PLOTted four orange dots, told you it was **COLOR 1, REG 0. ORANGE**, then PLOTted four green dots, and so forth. The computer doesn't think; it merely executes. You must do the thinking and mirror that thinking in your program.

To straighten out (debug) this program, we have to think the problem through. Obviously, the computer did what we requested, but too swiftly. We need to slow down the execution. How do we do that? We can make the computer do something, then pause and count to a certain number, do something else, count again, and so on. The easiest way to make the computer "count to itself" (or slow down its operation), is to use a **FOR...NEXT** loop. To add this to your program, just type it in at the end. Your computer will automatically put it in line number sequence.

```
80 FOR COUNT=1 TO 2000
90 NEXT COUNT
RUN
```

The first color part ran fine, but the second and third were still too fast. Type in the following to slow down the other color presentations and use the special trick.

```
160 FOR COUNT=1 TO 2000
170 NEXT COUNT
240 FOR COUNT=1 TO 2000
250 NEXT COUNT
RUN
```

Now everything works well, right? Well, not quite. We don't want the "READY" sign at the end of the program. Do you remember what to do to keep the screen from clearing? Type in:

```
260 GOTO 260
RUN
```

Now the program RUNs at a human speed, but the four blue dots stay on the screen at the end of the presentation. What we'd rather see, however, is the computer RUNning the program again and again. This is very simple to implement. Change line 260 to:

```
260 GOTO 10
RUN
```

Your Atari will continue to cycle through these colors as long as you allow it to. Press **BREAK**, and the program will stop.

If your program did not work properly, check your LISTing against the next one. Press the **SYSTEM RESET** key, type **LIST**, and press **RETURN**.

```
10 GRAPHICS 3
20 COLOR 1
```

7 GRAPHICS THREE, FIVE, AND SEVEN

```
30 PLOT 15,5
40 PLOT 25,5
50 PLOT 15,15
60 PLOT 25,15
70 ? "THIS IS COLOR 1,REG.0:ORANGE."
80 FOR COUNT=1 TO 2000
90 NEXT COUNT
100 COLOR 2
110 PLOT 15,5
120 PLOT 25,5
130 PLOT 15,15
140 PLOT 25,15
150 ? "THIS IS COLOR 2,REG.1:GREEN."
160 FOR COUNT=1 TO 2000
170 NEXT COUNT
180 COLOR 3
190 PLOT 15,5
200 PLOT 25,5
210 PLOT 15,15
220 PLOT 25,15
230 ? "THIS IS COLOR 3,REG.2:BLUE."
240 FOR COUNT=1 TO 2000
250 NEXT COUNT
260 GOTO 10
```

There's something else you can do to clean up the program. Type in the following:

```
95 GRAPHICS 3
175 GRAPHICS 3
RUN
```

Now you have only one message in the window at the bottom, which makes it easier to read the correct message. The GR.3 statement clears the screen.

As I said earlier, this program is one way to display the three different colors. Right now, let's shorten it. (Remember, to SAVE some of these programs.) Press the SYSTEM RESET key, type in NEW, and press the RETURN key. Type in the following program, please:

```
10 GRAPHICS 3
20 FOR C=1 TO 3
30 COLOR C
40 PLOT 20,10
50 PLOT 20,11
60 PLOT 20,12
70 PLOT 20,13
80 IF C=1 THEN ? "THIS IS COLOR 1,REGISTER
  0:ORANGE."
90 IF C=2 THEN ? "THIS IS COLOR 2,REGISTER
  1:GREEN."
100 IF C=3 THEN ? "THIS IS COLOR 3,REGISTER
```



```

        2:BLUE."
110 FOR COUNT=1 TO 2000
120 NEXT COUNT
130 GRAPHICS 3
140 NEXT C
150 GOTO 10
RUN

```

The program is lines shorter than the last one, and sometimes, depending on the amount of RAM a user has, that's important. At present, a line-by-line discussion of the program we just ran, would be most valuable.

- Line 10 Regular GR.3 with a TW.
- Line 20 Tells the computer that C has the values of 1, 2, and 3. This is the beginning of a FOR...NEXT loop that supplies line 30 with the color numbers.
- Line 30 When the computer reaches this line, it reads COLOR C, but it knows that C=1, 2, and 3. Since this is the first time through the loop, the computer will change COLOR C to COLOR 1. It will then go on to the next line.
- Lines 40-70 Instead of four unconnected dots making a "box," we have four connected ones making a vertical bar.
- Line 80 This convenient command in BASIC is the IF...THEN statement which causes the computer to compare one thing to another thing. In this case, it tells the Atari, "If C equals 1 (or, IF the COLOR C statement equals COLOR 1) THEN PRINT the message that follows." Since this is the first time through the loop, COLOR C *does* equal COLOR 1, therefore the message will be printed.
- Line 90 Again the computer comes to an IF...THEN statement; it is told to PRINT the message IF C=2. C does *not* equal two (it equals one). Therefore, the computer will *not* print the message; instead, it will go to line 100.
- Line 100 Once again, the computer must compare two things, then make a decision. The computer first asks if C equals 3. Once it has made this comparison, it must act. If the comparison is true (and C does equal 3), it will print the message. Since this is the first time through the program, C does *not* equal 3, the computer will *not* print the message, and goes on to the next line.
- Line 110-120 Here is our old friend, the pause loop. This time it's on two lines. (You can put it on one line if you wish.)
- Line 130 Clears the message from the TW. Also clears the screen, as well.
- Line 140 Tells the computer to go back to the beginning of the C loop (line 20) and change C until it equals 2, making the next line COLOR 2, which is green. It will then make the four dots a green color. This time through the loop, though, it ignores the PRINT message in

7 GRAPHICS THREE, FIVE, AND SEVEN

line 80 because C does not equal 1. It goes on to line 90, and sure enough, C *does* equal 2; it PRINTs the message in this line.

The Atari goes on to line 200 where it has to make another comparison. C doesn't equal 3, so it does not execute the PRINT instruction. Instead it goes on to the counting loop, pauses, then goes to line 130 where it is told to clear the TW and screen. Then it goes to line 140 for the NEXT C. The computer remembers that C was equal to 1, 2, and 3 (line 20), so it returns to line 20 again, makes C equal 3, then goes through the loop again, making a blue line.

Finally, it reaches line 140 again where it is told to do the NEXT C. It won't do the NEXT C because it knows that C was supposed to go only as high as 3 (line 20). The next logical (and numerical) step is for it to go on to the next line (150) which it does.

Line 150 The computer is told to go back to line 10. When it gets there, it will begin the program all over again and RUN the same program endlessly.

Let's talk a bit more here about FOR...NEXT loops. Notice that lines 20 and 140 go together. Or, to put it another way, line 140 is the "end" of the loop that begins with line 20. Notice the other loop on lines 110 and 120. Do you recall that it is called a nested loop because it's nested *inside* another loop? In this case, the count loop is completed *inside* the other loop. You could not, for example, change line 120 to line 135 and make the program work, error-free. It must complete the *interior* loop before it goes on to the *exterior* one.

Type in the following, please, after you type NEW to clear your computer:

```
10 FOR LIMITER=1 TO 1000
20 NEXT LIMITER
RUN
```

This program will make the computer count to 1000, then it will place the "READY" prompt on the screen. This is the simplest FOR...NEXT loop. Use NEW to erase old lines from your computer, and type in:

```
10 FOR LIMITER=1 TO 3
20 ? "LOVE IS GREAT!"
30 NEXT LIMITER
RUN
```

When you have a task for the computer, such as printing the LOVE IS GREAT! statement, the computer prints it the number of times specified in the limiter. The computer will, in this case, print *LOVE IS GREAT!* three times.

Use NEW to erase the old program from your machine, and type this in, please:

```
10 FOR LIMITER=1 TO 3
20 ? "LOVE IS GREAT!"
```

```
30 ? "BEAUTY IS TO BE APPRECIATED."
40 NEXT LIMITER
RUN
```

If there are multiple things for the computer to do between the beginning and the end of a single FOR...NEXT loop, the number of times it will do them is controlled by the initial statement in the FOR part of the FOR...NEXT loop. That's why the computer printed LOVE IS GREAT! and BEAUTY IS TO BE APPRECIATED once, a second and third time, then stopped.

Type this in, please. (I'm sure I needn't remind you to type in NEW, etc., each time.)

```
10 FOR LIMITER=1 TO 3
20 ? "HICKORY"
30 FOR SLOWDOWN=1 TO 200
40 NEXT SLOWDOWN
50 NEXT LIMITER
RUN
```

The computer sets the limiter to 1, but knows it must eventually be set to 2, then 3.

It prints **HICKORY**. Then it encounters another FOR...NEXT loop with limiters of 1 to 200, so whatever instructions follow must be done 200 times. Since there is nothing between the beginning of the SLOWDOWN loop and the end of it, the Atari will simply count to 200.

The next statement is the end of the limiter loop. The computer has two choices: it can go on to the next line, or revert to the beginning of that specific FOR...NEXT loop. Here, it must return to the beginning because it has to do whatever it started a total of three times, and it's only done it once.

The computer will go through the process a total of three times, then go on to the next line and follow the instructions there. Your loop ends when the initial FOR...NEXT loop runs out by reaching the last number in the limiter loop.

If you have twenty nested FOR...NEXT loops, they will be completed logically. When your computer encounters a NEXT statement, it goes back to its matching FOR limiter until it's done whatever it has to, for however many times the limiter tells it to.

Sometimes, FOR...NEXT loops seem to be capricious little creatures that enjoy themselves by disrupting your harmony as well as your programs. Really though, they don't have a mind of their own—nor does the computer. One is an instruction and the other is a machine. They're both dumb critters that simply do the jobs for which they were created. Unfortunately, the FOR...NEXT loops and the machine won't do what you *want* them to do; they only do what you *tell* them to, or indicate that they can't follow your instructions. The ball is then in your court.

7 GRAPHICS THREE, FIVE, AND SEVEN

Relax. You can always figure out what is wrong with your program no matter what the computer does.

Probably, you already know some or all of the things I'm going to go over, but it won't hurt to review them.

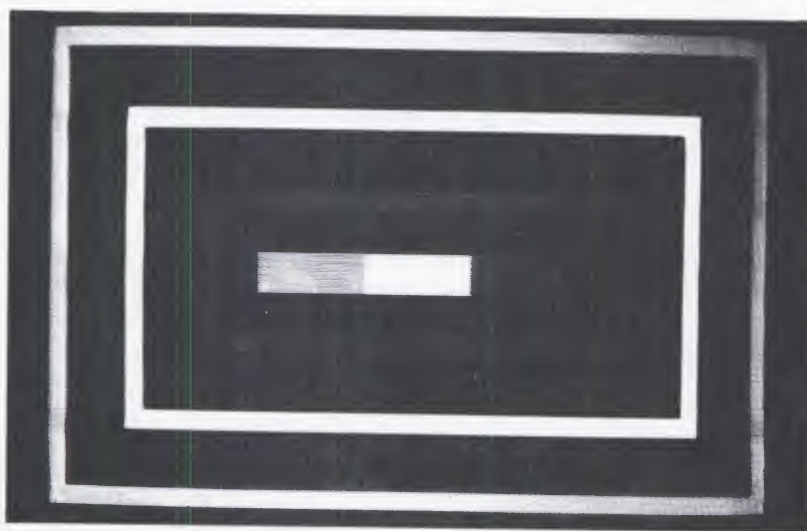
The abbreviation for the word SETCOLOR is **SE**.

The abbreviation for the word PLOT is **PL**.

The abbreviation for DRAWTO is **DR**. This command tells the computer to DRAW a line TO the X (column) and Y (row) coordinates you set. For example: DRAWTO 10,25 tells the computer to DRAW a line TO column 10 (X coordinate) and row 25 (Y coordinate).

And, the colon (:) is used to separate two or more commands on one line. It saves typing many line numbers, and saves RAM as well.

Use NEW and RETURN to erase old program lines. Use the trick, too, if you wish. The next program places three colors on the screen at once.



```
10 GRAPHICS 3+16
20 COLOR 1
30 PLOT 0,0:DRAWTO 39,0
40 DRAWTO 39,23:DRAWTO 0,23
50 DRAWTO 0,0
60 COLOR 2
70 PLOT 4,4:DRAWTO 4,19
80 DRAWTO 35,19:DRAWTO 35,4
90 DRAWTO 4,4
100 COLOR 3
110 PLOT 8,8:DRAWTO 31,8
120 DRAWTO 31,15:DRAWTO 8,15
```



```

130 DRAWTO 8,8
140 COLOR 1
150 PLOT 11,11:DRAWTO 16,11
160 PLOT 11,12:DRAWTO 16,12
170 COLOR 2
180 PLOT 17,11:DRAWTO 22,11
190 PLOT 17,12:DRAWTO 22,12
200 COLOR 3
210 PLOT 23,11:DRAWTO 28,11
220 PLOT 23,12:DRAWTO 28,12
400 GOTO 400
RUN

```

You should have three boxes, nested one inside the other. The outer box is orange, the middle box green, and the innermost box blue. Inside the blue box is one rectangle of orange, green, and blue.

For the moment, let's assume you don't like COLOR 1. Can you change it? Of course. COLOR 1 is controlled by register 0, so you use SETCOLOR 0, COLOR, BRIGHTNESS. Change line 20 to the following:

```

20 COLOR 1:SETCOLOR 0,12,2
RUN

```

You now have a green color for the outside box instead of orange. Notice how it affects the other colors. Pretty, isn't it?

Let's change COLOR 2 while we're at it. Remember, COLOR 2 is controlled by register 1, so you use the word SETCOLOR (or SE.) with the register number.

```

60 COLOR 2:SETCOLOR 1,3,2
RUN

```

Now you have a green and a red box. Let's change COLOR 3; it is controlled by register 2.

```

100 COLOR 3:SETCOLOR 2,6,0
RUN

```

You have green, red, and blue boxes. If you wish to change the background and the border, you use SETCOLOR 4,COLOR,BRIGHTNESS. SE.4,9,12 changes the border and background to a pale blue (on my TV set), and changes the other colors, too. Experiment to get the colors that please you most. Please add:

```

10 GRAPHICS 3+16:SETCOLOR 4,9,12
RUN

```

Hope you like it. If not, feel free to make changes. The more your practice, the more adept you will become at using color.

7 GRAPHICS THREE, FIVE, AND SEVEN

Let's talk a minute about putting a *block* of color on the screen. There are several ways to do this. First of all, you could PLOT and DRAWTO as you did in lines 150 and 160, 180 and 190, and 210 through 220.

Another way to accomplish this is to use our comrade, the FOR...NEXT loop. SYSTEM RESET and type NEW to clear your computer, please. Remember, color works the same in GR.3, 5, and 7. Type in:

```
10 GRAPHICS 7+16
20 COLOR 1
30 FOR X=70 TO 90
40 FOR Y=30 TO 50
50 PLOT X,Y
60 NEXT Y
70 NEXT X
100 GOTO 100
RUN
```

This is a neat way to draw a shape on your screen. How does it work? The key is the FOR...NEXT loop. To see how it works, put in a counting loop to slow down the execution of the program. Add this, please:

```
55 FOR WAIT=1 TO 99:NEXT WAIT
RUN
```

The letter "X" refers to the column number, and the "Y" to the row number. When you want to place something on the screen at a specific location, put the column number first, followed by the row number.

You've told the computer that X equals 70 and Y equals 30 the first time through the loop. So when the computer reaches line 50, it puts a dot of color at 70,30 as if the line had said PL.70(X), 30(Y).

Next, it counts to 99. After this, it goes to line 60 where it is told to do the NEXT Y. Since Y=30 TO 50 and you've already used 30, the computer makes Y=31 and places a dot at coordinates 70,31. It continues and will run through all the remaining Y coordinates (32,33,34, etc.) until it reaches 70,50.

It then does the NEXT X by going to line 30 and making X equal 71. Next, the computer goes to line 40 and makes Y equal 30 again. The computer then starts to PLOT in column 71 and does what was done in 70: it puts a dot of color in each Y coordinate (31, 32, 33, 34, etc.) until it reaches coordinates 71,50. Then it goes back to line 30, makes X=72, and starts again with Y=30.

The program finishes when the computer has put a dot in the twenty-one columns (70 through 90, X coordinates), and in the twenty-one rows (30 through 50, Y coordinates). If you'd like to see this more clearly, change the FOR...NEXT loop in line 55 to:

```
55 FOR WAIT=1 TO 200:NEXT WAIT
RUN
```


Study the program until it becomes clear to you. Slowing its execution down is a great help. Practice using this new application of the FOR...NEXT loop until you can put a block of color on the screen wherever you want it.

Another way to color in an area is by using the XIO statement. Please type in the following program after you press SYSTEM RESET and again use NEW to clear your computer. You will be placing three colored boxes on the screen; use the trick.

```

100 REM USING THE XIO FILL ROUTINE
110 GRAPHICS 7+16
120 COLOR 1
130 PLOT 20,20:DRAWTO 20,0
140 DRAWTO 0,0
150 POSITION 0,20
160 POKE 765,1
170 XIO 18,#6,0,0,"S:"
200 COLOR 2
210 PLOT 90,56:DRAWTO 90,36
220 DRAWTO 70,36
230 POSITION 70,56
240 POKE 765,2
250 XIO 18,#6,0,0,"S:"
300 COLOR 3
310 PLOT 159,91:DRAWTO 159,70
320 DRAWTO 140,70
330 POSITION 140,91
340 POKE 765,3
350 XIO 18,#6,0,0,"S:"
500 GOTO 500
RUN

```

You should have three colored squares: an orange one in the upper left-hand corner, a green in the middle, and a blue in the lower right-hand corner. LIST your program.

I divided the program into multiples of 100 because each unit of 100 contains the instructions for one box. It's easiest to discuss one unit only. Except for the colors, they all work the same.

- Line 200 Use green paint.
- Line 210 Start your PLOTting at the lower right-hand corner of your figure. DRAWTO the upper right-hand corner.
- Line 220 DRAWTO the upper left-hand corner of the figure.
- Line 230 The fourth corner of your figure should be the lower left-hand corner, since you've drawn a square or a rectangle.
- Line 240 This is the first part of the XIO fill statement. Always put POKE 765, followed by the number of the COLOR statement you are using.

7 GRAPHICS THREE, FIVE, AND SEVEN

Here you should have POKE 765,2 since you are using COLOR 2 (line 200).

Line 250 Use this same formula every time and the computer will fill your figure with color.

We did three squares and filled them with color. Can we do the same for a triangle? This time, there'll be only 1 PLOT, 1 DRAWTO, and 1 POSITION. Let's give it a try. SAVE the program, if you wish, then use NEW and RETURN to erase old program lines.

```
10 GRAPHICS 7+16
20 COLOR 2
30 PLOT 98,40:DRAWTO 80,22
40 POSITION 62,40:REM YOU MUST POSITION
   YOUR FINAL COORDINATE.
50 POKE 765,2
60 XIO 18,#6,0,0,"S:"
70 GOTO 70
```

Worked fine, didn't it? Let's change the color using SETCOLOR and see what happens. Add this to your program:

```
20 COLOR 2:SETCOLOR 1,10,8
RUN
```

Now you have a nice aqua color. But if you cut the triangle in half, will it still work? Make this change:

```
40 POSITION 80,40
RUN
```

It still works fine. (I plotted these triangles beforehand on a piece of graph paper so I knew they'd be perfect.) What would happen if you made the POSITION 83,40? Try it.

```
40 POSITION 83,40
RUN
```

Now you have a strange-looking triangle. Make the POSITION 78,40.

```
40 POSITION 78,40
RUN
```

Is it still strange-looking? You need to use *straight lines* in your triangle, rectangle, or square. If you don't, you'll still get the "fill" activity, but it will be awkward-looking.

Try a host of different figures using the XIO. Some will work, others won't. Be creative and explore, adding more tricks to your programming bag.

One crucial piece of information remains: the XIO fill activity will be disturbed if you have previously put a dot of color inside the area you want to fill.

Use the following program as the basis for four assignments (later, you'll see how I solved them). Remember, though, as long as your program does what mine does, you're okay. Our line numbers probably won't match, but don't worry about that.

The program draws a box. To allow you enough room for changes, I've written the line numbers in multiples of 100. Type in the following:

```
100 GRAPHICS 7+16
200 COLOR 2
300 PLOT 105,40:DRAWTO 105,20
400 DRAWTO 55,20:POSITION 55,40
500 POKE 765,2
600 XIO 18,#6,0,0,"S:"
700 GOTO 700
RUN
```

Assignment #1

Change the color in line 200 by using SETCOLOR. Remember that COLOR 2 is controlled by register 1. (Please leave each change you make on these four assignments.)

Assignment #2

Have your computer cycle through the sixteen colors. You'll have to build your FOR...NEXT loop around the SETCOLOR statement. Use the variable COLOR for the FOR...NEXT statement. After each assignment, RUN the program and see if it works; if it doesn't, debug it.

Assignment #3

That was too fast. Slow the changes down. Remember to nest the FOR...NEXT loops.

Assignment #4

Have the computer cycle through the eight brightnesses. Since the brightnesses go from 0 to 14 (even numbers only), put STEP 2 after the FOR statement. Use the variable LUM.

Solution #1

```
200 COLOR 2:SETCOLOR 1,15,6
```

On my TV, the color changes from green to orange; I doubt that our color choices will match.

Solution #2

```
150 FOR COLOR=0 TO 15
200 COLOR 2:SETCOLOR 1,COLOR,6
650 NEXT COLOR
```

Put in the FOR...NEXT loop *before* the XIO business starts, and *after* it is finished.

7 GRAPHICS THREE, FIVE, AND SEVEN

Solution #3

```
625 FOR WAIT=1 TO 150:NEXT WAIT
```

You should have placed this *after* the XIO statement and *before* the NEXT COLOR statement.

Solution #4

```
160 FOR LUM=0 TO 14 STEP 2
200 COLOR 2:SETCOLOR 1,COLOR,LUM
```

As long as you change the SETCOLOR statement and as long as your computer picks one color, RUNs through the eight brightnesses, picks another color and so forth, until it has done the sixteen colors with eight brightnesses for each one, you are right.

The following is my final LISTing. The previous assignments determined if you could do the different tasks. If you couldn't, don't be upset. (While I was making up this program, I forgot to change the SETCOLOR statement by adding the variable COLOR. It took me a while to debug the program.)

If you made all the changes without looking at the solutions, I'm impressed. You're doing a super job!

Finally, here's the last LISTing:

```
100 GRAPHICS 7+16
150 FOR COLOR=0 TO 15
175 FOR LUM=0 TO 14 STEP 2
200 COLOR 2:SETCOLOR 1,COLOR,LUM
300 PLOT 105,40:DRAWTO 105,20
400 DRAWTO 55,20:POSITION 55,40
500 POKE 765,2
600 XIO 18,#6,0,0,"S:"
625 FOR WAIT=1 TO 150:NEXT WAIT
650 NEXT LUM
675 NEXT COLOR
700 GOTO 700
```

As long as our programs do the same things, you are fine. If you have the time, why not have the background color cycle through all the colors? If you're feeling even more ambitious, you could make the background cycle through all the colors *and* brightnesses.

The following program will draw a picture of a wine goblet or chalice. I'm not crazy about the colors, and I don't think you will be, either. Feel free to change them until you get colors that please you. You should use the trick on this one since the lines are so similar.



```

10 GRAPHICS 5+16
20 COLOR 1
30 REM DO THE TRICK
40 REM BOTTOM
50 PLOT 32,45:DRAWTO 48,45
60 PLOT 33,44:DRAWTO 47,44
70 PLOT 34,43:DRAWTO 46,43
80 PLOT 35,42:DRAWTO 45,42
90 PLOT 36,41:DRAWTO 44,41
100 PLOT 37,40:DRAWTO 43,40
110 PLOT 38,39:DRAWTO 42,39
115 COLOR 2
118 REM MIDDLE
120 PLOT 39,38:DRAWTO 41,38
130 PLOT 39,37:DRAWTO 39,15
140 PLOT 40,37:DRAWTO 40,15
150 PLOT 41,37:DRAWTO 41,15
160 PLOT 38,14:DRAWTO 42,14
170 PLOT 39,13:DRAWTO 39,11
180 PLOT 40,13:DRAWTO 40,11
190 PLOT 41,13:DRAWTO 41,11
195 REM TOP
200 PLOT 38,10:DRAWTO 42,10
210 PLOT 37,9:DRAWTO 43,9
220 PLOT 36,8:DRAWTO 44,8
230 PLOT 35,7:DRAWTO 45,7
240 PLOT 34,6:DRAWTO 46,6
250 PLOT 33,5:DRAWTO 47,5
260 PLOT 32,4:DRAWTO 48,4
270 PLOT 31,3:DRAWTO 49,3
280 PLOT 30,2:DRAWTO 50,2
290 PLOT 29,1:DRAWTO 51,1

```

7 GRAPHICS THREE, FIVE, AND SEVEN

```
300 PLOT 28,0:DRAWTO 52,0
400 GOTO 400
RUN
```

See what I mean about the colors? Work on them, then SAVE the program and/or erase old program lines from your computer before beginning the next program.

The following programs create some interesting drawings and give you more practice with FOR...NEXT loops, the STEP statement, and the IF...THEN statement.

The next program will draw an X:

```
10 GRAPHICS 5+16:REM GR.3,5,AND 7 WORK THE SAME
   FOR COLOR
20 FOR B=0 TO 14 STEP 2
30 COLOR 3:SETCOLOR 2,8,B
40 PLOT 0,0:DRAWTO 79,39
50 PLOT 79,0:DRAWTO 0,39
60 FOR WAIT=1 TO 300:NEXT WAIT
70 NEXT B
80 GOTO 80
RUN
```

With the pause loop in line 60, the execution of the program is slow enough for us to examine what is happening. If you count the number of brightness changes, you'll see that there are eight—one for each time through the B loop. What we have is COLOR 8, BRIGHTNESS 0: COLOR 8, BRIGHTNESS 2, and so on until we reach COLOR 8, BRIGHTNESS 14. The STEP statement can be quite a handy tool for cycling through the colors.

Make the following changes, please:

```
20 FOR T=0 TO 3
30 COLOR T
70 NEXT T
RUN
```

Now you see the same X, but it cycles through COLOR 0, COLOR 1, COLOR 2, and COLOR 3. The X should be invisible or black (COLOR 0), orange (COLOR 1), green (COLOR 2), and blue (COLOR 3).

It looks different because the computer actually draws the X in black, the background color, then changes the color to orange. That's why the orange X seems so abrupt.

Next, change line 20, please:

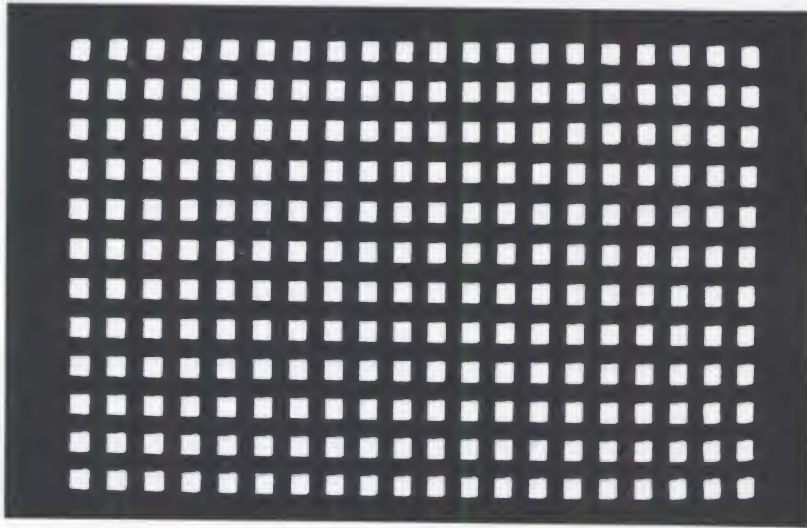
```
20 FOR T=1 TO 3 STEP 2
RUN
```

The computer makes T equal 1 (orange); then it skips T equals 2 (because of the STEP 2 statement); then it makes T equal 3, which is blue.

If you wish, add the following to make the program go on and on:

```
80 GOTO 10
```

You should see the color change from orange to blue, to orange, and so on.



Let's do another program using the STEP 2 statement that will make a green and black checkerboard. Press SYSTEM RESET, then type NEW, and press RETURN.

```
10 GRAPHICS 3+16
```

Makes screen 40 x 24.

```
20 COLOR 2
```

Green.

```
30 FOR X=0 TO 38 STEP 2
```

There are 40 columns (X), numbered 0 to 39. X will equal 0, then 2, 4, etc.

```
40 FOR Y=0 TO 22 STEP 2
```

There are 24 rows (Y), numbered 0 to 23. That's why we can't have Y=0 TO 24. Y will equal 0, then 2, 4, etc.

```
50 PLOT X,Y
```

Put a green dot at X(0) and Y(0).

```
60 NEXT Y
```


7 GRAPHICS THREE, FIVE, AND SEVEN

Y did equal 0. Now make it 2, then 4, 6, and so on until *Y* equals 22.

The computer should put a green dot at 0,0. It then should put a green dot at 0,2; then 0,4, continuing in this fashion until it places a green dot at 0,22 (column 0, row 22). Every other spot in column 0 should be black, the background color.

```
70 NEXT X
```

After *Y* equals 22, the computer goes on to this line which sends it back to line 30 and makes *X* equal 2. The computer places dots in column 2, just as it did in column 0, meaning it places a dot at column 2, row 0, then at column 2, row 2, etc., until it reaches column 2, row 22.

```
80 GOTO 80  
RUN
```

You have created a nice green and black checkerboard. If you'd like to slow down the presentation, put in a pause loop at line 55:

```
55 FOR WHOA=0 TO 25:NEXT WHOA  
RUN
```

STEP 2 can be helpful for drawings. Study this program if you're not sure how the STEP works. Make the pause loop even slower, for practice.

Let's use the STEP 2 to cycle through the three graphics modes: 3, 5, and 7. Add this to your program, please (type these at the bottom of your LISTing). The first line 10 will be deleted and this one will take its place:

```
5 FOR MODE=3 TO 7 STEP 2  
10 GRAPHICS MODE+16  
75 NEXT MODE  
RUN
```

This program should have been presented to you in GRAPHICS 3 mode, then in GRAPHICS 5, and in GRAPHICS 7. The MODE will equal 3, then 5, then 7 because of the STEP 2 statement in line 5. Another interesting way to use the STEP 2 and FOR...NEXT loop is in reverse. Change line 30 as follows:

```
30 FOR X=38 TO 0 STEP-2:REM CAUTION. IF YOU USE A NEGATIVE  
STEP, THE FIRST NUMBER MUST BE THE HIGHEST (38 TO 0,  
NOT 0 TO 38)  
RUN
```

This time the checkerboard starts in the upper right-hand corner instead of the upper left. It places dots in column 38 in every other row; it does the same in column 36, then 34, etc., until it completes column 0 as well, due to the negative STEP.

Can you start the activity in the lower right-hand corner? Make this change, please:

```
40 FOR Y=22 TO 0 STEP -2  
RUN
```

Your computer should place a dot of color at coordinates 38,22; 38,20 and so on, until it finishes column 0 with a dot of color at 0,0.

You can also have the computer start in the lower left-hand corner. Make these changes, please:

```
30 FOR X=0 TO 38 STEP 2
40 FOR Y=22 TO 0 STEP -2
RUN
```

The computer did what you asked it to do, right? If not, check your LISTing against the following one:

```
5 FOR MODE=3 TO 7 STEP 2
10 GRAPHICS MODE+16
20 COLOR 2
30 FOR X=0 TO 38 STEP 2
40 FOR Y=22 TO 0 STEP -2
50 PLOT X,Y
55 FOR WHOA=0 TO 25:NEXT WHOA
60 NEXT Y
70 NEXT X
75 NEXT MODE
80 GOTO 80
RUN
```

If the presentation is too slow for you, remove the pause loop. You can also make the colors cycle in each graphics mode by typing in:

```
8 FOR CHANGE=1 TO 3
20 COLOR CHANGE
72 NEXT CHANGE
RUN
```

There are many other things you can do with the judicious use of FOR...NEXT loops and STEP statements. Let's combine these two with an IF...THEN statement and see what happens. Delete lines 8 and 72, type an 8, and press RETURN. Type 72, press RETURN, and add:

```
20 COLOR 1
56 IF X=12 AND Y=0 THEN COLOR 2
57 IF X=24 AND Y=0 THEN COLOR 3
RUN
```

This is a variation of the IF...THEN statement. I call it the **IF AND THEN** statement. It tells the computer to do something if *two* things are true. For example, **IF** the column number is 12 **AND** the row number is zero, **THEN** change the color to 2. **IF** the column number is 24 **AND** the row number is 0, **THEN** change the color to 3.

Every other column still has a dot of color in every other row, but this time

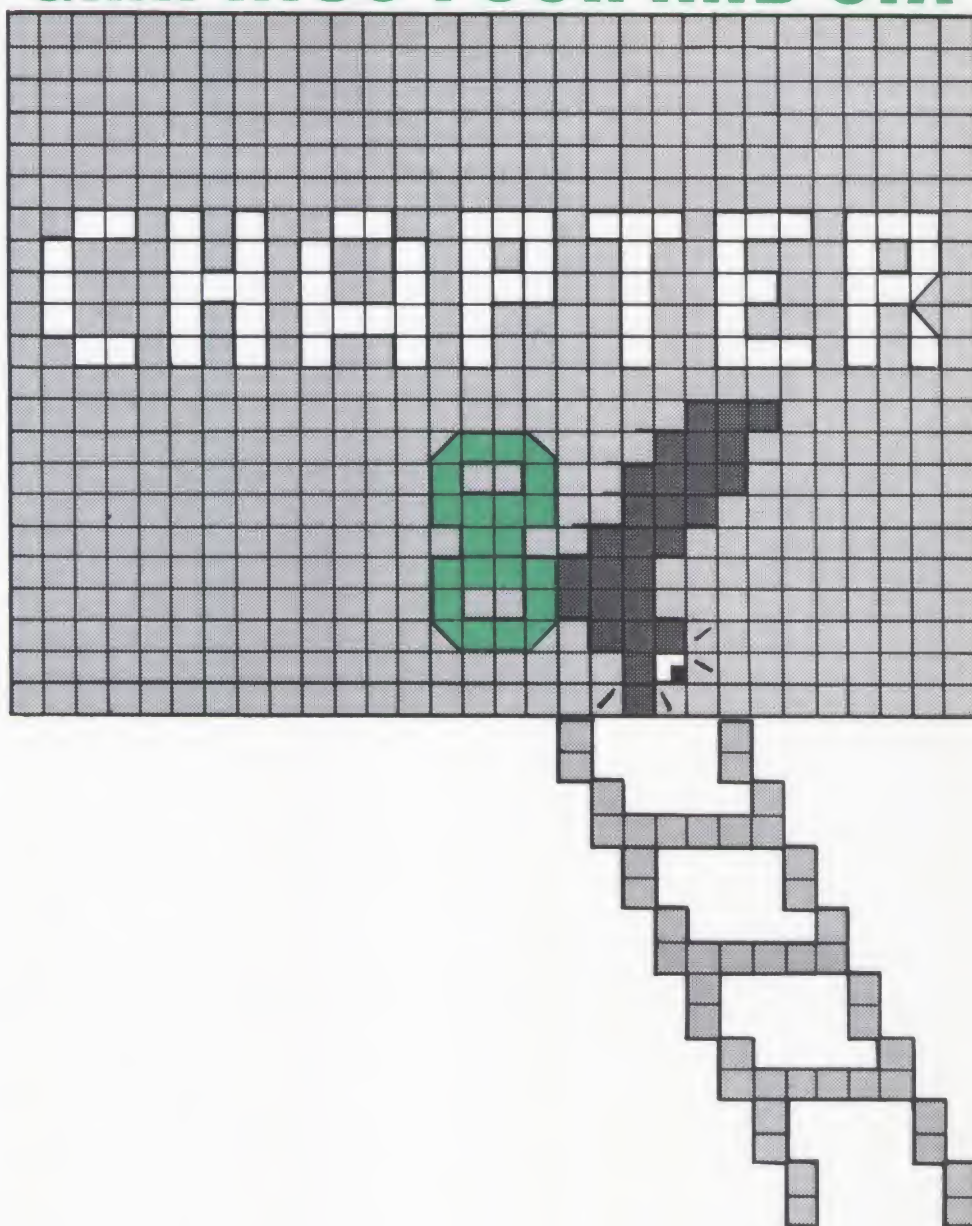
7 GRAPHICS THREE, FIVE, AND SEVEN

three colors are displayed: COLOR 1 for columns 0 through 10; COLOR 2 for columns 12 through 22; and COLOR 3 for columns 24 through 38.

One change you could try would be changing the color when you switch the graphics modes. Experiment freely with GR.3, 5, and 7; or go back to GR.0, 1, and 2 for a review. If, however, you comprehend and are comfortable with what I've presented thus far, let's proceed to GRAPHICS 4 and 6, which you'll find are very simple.



GRAPHICS FOUR AND SIX





Graphics Four and Six

The modes below are two-color modes. There is one color for the border/background which is controlled by `SETCOLOR 4,COLOR,BRIGHTNESS`; there also is one color for the foreground which is controlled by `COLOR 1` and `SETCOLOR 0,COLOR,BRIGHTNESS`.

Mode	Screen Positions
GRAPHICS 4	80 by 40
GRAPHICS 4+16	80 by 48
GRAPHICS 6	160 by 80
GRAPHICS 6+16	160 by 92

The following program has five purposes. The first is to demonstrate that the presentation of a drawing can sometimes be more important than the drawing itself. There is just one line being drawn, but the *way* it is drawn involves the observer.

The second purpose demonstrates the use of the `GOSUB` statement. The third shows you that these two modes can help simulate depth because the vertical (top to bottom) lines are a bit darker than the horizontal (side to side) lines.

The fourth purpose teaches you how to use color in these modes, and the fifth gives you more practice using the special trick (since you will have to change only one or two digits in most program lines). Using the trick makes this rather long program simple to type:

```
10 GRAPHICS 4+16
20 SETCOLOR 4,8,8:REM BORDER/BACKGROUND
30 COLOR 1
```


8 GRAPHICS FOUR AND SIX

```
40 SETCOLOR 0,14,14:REM PLOTTING COLOR
50 PLOT 40,24:GOSUB 540
60 DRAWTO 40,22:GOSUB 540
70 DRAWTO 38,22:GOSUB 540
80 DRAWTO 38,26:GOSUB 540
90 DRAWTO 42,26:GOSUB 540
100 DRAWTO 42,20:GOSUB 540
110 DRAWTO 36,20:GOSUB 540
120 DRAWTO 36,28:GOSUB 540
130 DRAWTO 44,28:GOSUB 540
140 DRAWTO 44,18:GOSUB 540
150 DRAWTO 34,18:GOSUB 540
160 DRAWTO 34,30:GOSUB 540
170 DRAWTO 46,30:GOSUB 540
180 DRAWTO 46,16:GOSUB 540
190 DRAWTO 32,16:GOSUB 540
200 DRAWTO 32,32:GOSUB 540
210 DRAWTO 48,32:GOSUB 540
220 DRAWTO 48,14:GOSUB 540
230 DRAWTO 30,14:GOSUB 540
240 DRAWTO 30,34:GOSUB 540
250 DRAWTO 50,34:GOSUB 540
260 DRAWTO 50,12:GOSUB 540
270 DRAWTO 28,12:GOSUB 540
280 DRAWTO 28,36:GOSUB 540
290 DRAWTO 52,36:GOSUB 540
300 DRAWTO 52,10:GOSUB 540
310 DRAWTO 26,10:GOSUB 540
320 DRAWTO 26,38:GOSUB 540
330 DRAWTO 54,38:GOSUB 540
340 DRAWTO 54,8:GOSUB 540
350 DRAWTO 24,8:GOSUB 540
360 DRAWTO 24,40:GOSUB 540
370 DRAWTO 56,40:GOSUB 540
380 DRAWTO 56,6:GOSUB 540
390 DRAWTO 22,6:GOSUB 540
400 DRAWTO 22,42:GOSUB 540
410 DRAWTO 58,42:GOSUB 540
420 DRAWTO 58,4:GOSUB 540
430 DRAWTO 20,4:GOSUB 540
440 DRAWTO 20,44:GOSUB 540
450 DRAWTO 60,44:GOSUB 540
460 DRAWTO 60,2:GOSUB 540
470 DRAWTO 18,2:GOSUB 540
480 DRAWTO 18,46:GOSUB 540
490 DRAWTO 62,46:GOSUB 540
500 DRAWTO 62,0:GOSUB 540
510 DRAWTO 18,0:GOSUB 540
520 PLOT 18,1
530 GOTO 530
```

```
540 FOR X=1 TO 300:NEXT X:RETURN
RUN
```

I think the pause after each line makes the presentation more interesting. The pause is accomplished by using the GOSUB statement. A GOSUB statement tells the computer to GO to a SUBroutine, which in this case is line 540. For instance, line 100 tells the computer to draw a line, then GO to line 540 where there is a SUBroutine that contains a standard WAIT loop followed by the word RETURN. The computer will DRAWTO 42,20, then go to line 540 where it counts to 300, then it will RETURN to the line following the GOSUB statement. In this case, it leaves line 540 and RETURNS to line 110. The RETURN causes the computer to RETURN to the next statement after the GOSUB. If, for example, there were a colon (:) after the GOSUB 540 and more instructions, the computer would RETURN to the statement after the GOSUB. The GOSUB statement helps us because we don't have to type a WAIT loop after every DRAWTO statement. You'll learn more uses for the GOSUB, later.

One thing you should notice is that the two shades of yellow give this drawing a feeling of depth (perspective). Notice, too, that the border/background is controlled by SETCOLOR 4,COLOR,BRIGHTNESS, and the drawing color is controlled by COLOR 1, which can be changed by using SETCOLOR 0,COLOR, BRIGHTNESS.

GRAPHICS 6 works in exactly the same way for color as GR.4, except it has twice as many pixels both horizontally and vertically.

Type in this picture of "The Professor." You'll see there are a number of places where you can use the trick to shorten your typing time.

```
10 GRAPHICS 6+16
20 SETCOLOR 0,12,8:COLOR 1:REM PLOTTING COLOR
30 SETCOLOR 4,9,2:REM BORDER/BACKGROUND COLOR
40 REM MORTAR BOARD
50 PLOT 71,7:DRAWTO 89,7:DRAWTO 92,9:PLOT 93,10
60 PLOT 74,7:DRAWTO 74,13
70 PLOT 86,7:DRAWTO 86,13
80 REM OUTLINE OF BODY, BEGINNING AT LEFT
  UPPER CHEEK
90 PLOT 71,16:DRAWTO 80,10
100 DRAWTO 89,16
110 DRAWTO 89,22:DRAWTO 83,28:DRAWTO 83,34
120 DRAWTO 98,34:DRAWTO 98,55:DRAWTO 89,64
130 DRAWTO 71,64:DRAWTO 62,55:DRAWTO
    62,34:DRAWTO 77,34
140 DRAWTO 77,28:DRAWTO 71,22:DRAWTO 71,16
150 REM RIGHT LEG
160 PLOT 83,64:DRAWTO 83,84:DRAWTO 87,84
170 REM LEFT LEG
```

8 GRAPHICS FOUR AND SIX

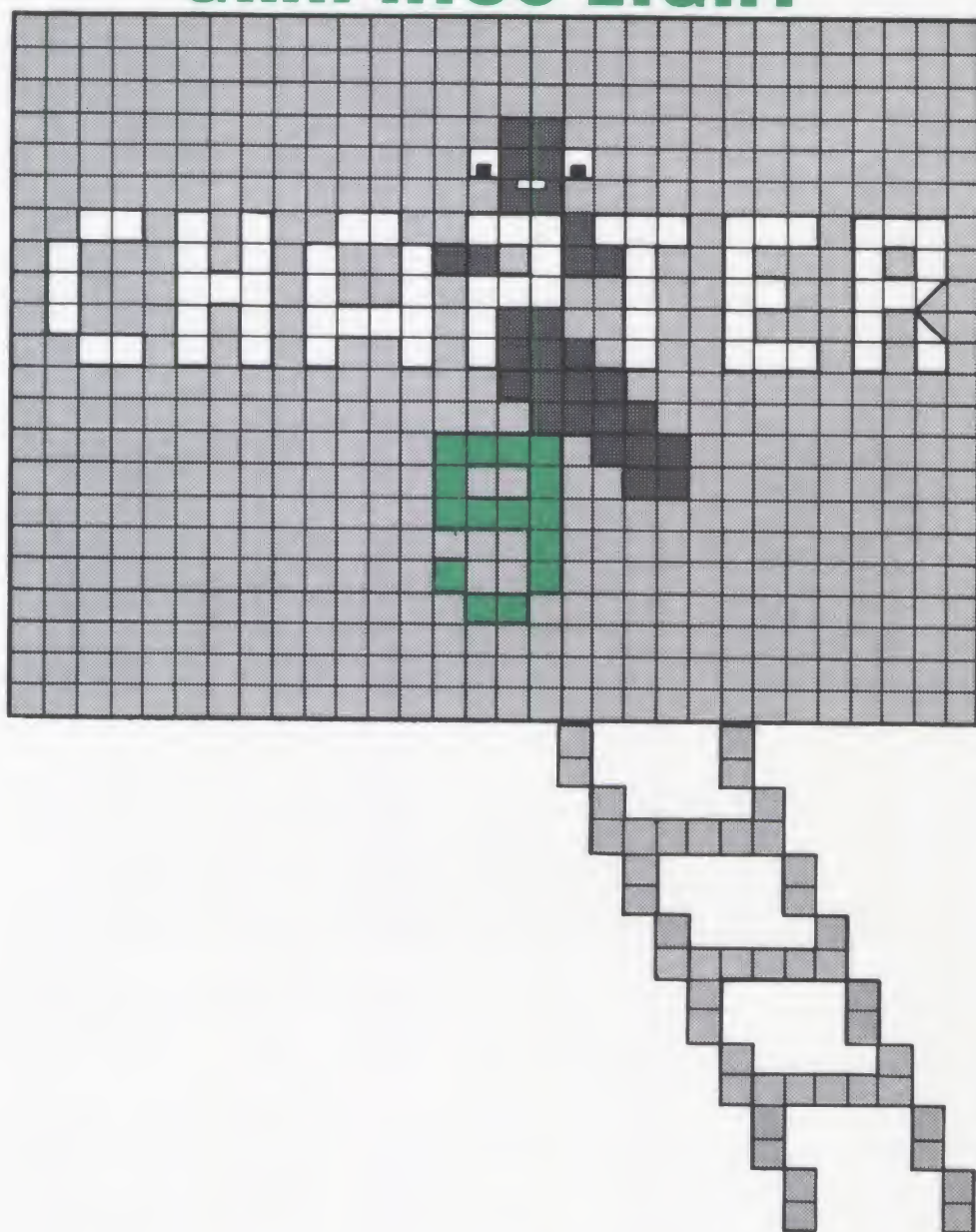
```
180 PLOT 77,64:DRAWTO 77,84:DRAWTO 73,84
190 REM RIGHT ARM
200 PLOT 98,46:DRAWTO 110,40:DRAWTO 110,30
210 REM LEFT ARM
220 PLOT 62,46:DRAWTO 50,40:DRAWTO 50,30
230 REM EYES
240 PLOT 76,16:DRAWTO 79,16:PLOT 81,16:
    DRAWTO 84,16
250 PLOT 77,15:DRAWTO 79,15
260 PLOT 81,15:DRAWTO 83,15
270 PLOT 78,14:PLOT 82,14
280 REM MOUTH
290 PLOT 78,23:DRAWTO 80,25
300 DRAWTO 82,23
310 REM NOSE
320 PLOT 79,18:DRAWTO 82,18:PLOT 80,19:
    DRAWTO 81,19
330 GOTO 330
RUN
```



This is enough to give you an idea of GRAPHICS 4 and 6. The good thing about GR.6 is that you get fairly high resolution (columns and rows) without using as much K as GR.8 (we'll talk about K shortly).

The negative side is that you have only two colors at your disposal: one for the foreground and one for the background/border. *Remember*, you can use the TW in these modes to interact with the user. Now on to GRAPHICS 8.

GRAPHICS EIGHT





Graphics Eight

Before we get into programming in GR.8, you should learn about the memory of your computer.

Remember when you first became interested in computers and talked to different salesmen? If they were anything like the ones I talked to, they rambled on about ROM and RAM. These two terms refer to the memory of your computer.

ROM or Read-Only Memory, is the memory area that contains the Operating System of your computer. This is the part that interprets your letters, words, and side (in a row) that makes one byte. From this, you can see it takes 64 bits, or 8 bytes in graphics mode, to form a symbol, letter, or number on the screen. Each K of RAM comprises 1,024 bytes. It is more complicated than this, but for the sake of

RAM, or Random Access Memory, is a second memory area, the place where you store your programs. Essentially, it is a bunch of boxes storing the letters, numbers, and symbols you type into your computer. RAM accepts your program and stores it until you type NEW and press RETURN, or until you turn off your machine.

Some overlap occurs here because the Operating System (OS) needs part of your RAM for instructions. In the graphics modes we've covered so far, the OS did not use much of your RAM. In GRAPHICS 8, the OS uses quite a bit of your RAM which can affect your programming—especially if you have only 16K of RAM. K stands for kilobytes or 2^{10} , which means approximately 1,000 bytes (though each K actually is 1,024 bytes). Remember the grid we used for the letter "A"? Recall that each box in the grid is stored in memory as a bit; eight of these bits placed side by side (in a row) makes one byte. From this, you can see it takes 64 bites, or 8 bytes in graphics mode, to form a symbol, letter, or number on the screen. Each K of RAM comprises 1,024 bytes. It is more complicated than this, but for the sake of

9 GRAPHICS EIGHT

simplification, we'll say that each K (1,024 bytes) holds 128 letters, numbers, or symbols. If you have only 3 K in this simplified scheme, you have room for 384 characters (128×3).

The computer's OS uses nearly 3K of your RAM just setting the machine up for programming in GR.8. Right now, turn on your computer and type this in, please:

```
PRINT FRE(0)
```

(Now type)

```
10 GRAPHICS 8  
RUN
```

The READY sign appears in the TW at the bottom of your screen. In this window, type:

```
PRINT FRE(0)
```

It takes nearly 7K just to begin GR.8. When I turned on my computer, the first thing I did was type in PRINT FRE(0). The number that came on my screen was 29,710. Theoretically, my computer has 32K; the Operating System uses nearly 3K to get ready when you turn on your computer.

I then typed in 10 GR.8, ran it, and typed in PRINT FRE(0) in the TW. The figure that came on the screen was 22,577, which means I lost 7,133 bytes, or nearly 7K. Just starting my machine and putting it into GR.8 cost me nearly 10K! You can readily see that you may have problems if you've got a long program in GR.8 and have only a 16K computer.

When you bought your 16 or 32K system, you may well have assumed you were purchasing 16 or 32K. Not so. PRINT FRE(0) will tell you how much K you really have.

GRAPHICS 8 is the highest resolution mode on the Atari, meaning it has the greatest number of pixels (picture cells) you can illuminate. There's a total of 51,200 boxes in GR.8 (320 columns by 160 rows), and 61,400 boxes in GR.8+16 (320 columns by 192 rows) wherein you can "be a Seurat" and use dots of color to draw a picture.

Georges Seurat (1859-1891) was the father of the *pointillistic* technique in art. He touched the paint on his brush to the canvas, creating a point on it. He repeated the process over and over again until his painting was finished. Seurat created many point-by-point detailed drawings, thus the terms *pointillistic* technique, and *pointillism*. In the different modes, as you know, you don't always have to draw by using single dots of color. With the XIO routine, you can fill the entire screen swiftly. But if you so desired, you *could* do a dot-by-dot drawing. You'll soon see why you might want to follow in the steps of Georges Seurat.

In GR.8, each pixel is half a color clock wide, which means two pixels side-by-side give you one color. However, if you illuminate only *one* of the pixels, you'll get a different color. In effect, if you illuminate only the odd-numbered pixels, you'll get one color; if you illuminate the even-numbered pixels, you'll get a different color; if you turn on an even pixel next to an odd pixel, you'll get a third color.

SUMMARY

- 1) odd-numbered pixels only = one color (Color A)
- 2) even-numbered pixels only = one color (Color B)
- 3) both odd- and even-numbered pixels side-by-side = one color (Color C)

Add the above to one color for the background and one color for the border, and you have a total of five colors. The term used to describe this unusual approach to placing color on the screen is **artifacting**.

You get one color and two brightnesses (as in GR.0). Use COLOR 1, SETCOLOR 1 for the foreground (your painting color), SETCOLOR 2 for the background, and SETCOLOR 4 for the border. Let's see how color works in this mode.

The next program shows you how to get three foreground colors in this one-foreground color mode.

```

10 GRAPHICS 8:POKE 752,1
20 COLOR 1
30 SETCOLOR 1,14,0
40 SETCOLOR 2,14,12
50 SETCOLOR 4,14,12
60 ? ".....XIO FILL ROUTINE....."
70 ?
80 ? ".....COLOR ALL PIXELS....."
90 FOR PAUSE=1 TO 350:NEXT PAUSE
100 POKE 765,1
110 PLOT 184,30:DRAWTO 184,15:DRAWTO 135,15
120 POSITION 135,30
130 XIO 18,#6,0,0,"S:"
140 FOR PAUSE=1 TO 350:NEXT PAUSE
150 ? "]"
160 FOR PAUSE=1 TO 350:NEXT PAUSE
170 ? ".....FOR-NEXT LOOP....."
180 ? ".....COLOR PIXELS IN EVERY....."
190 ? ".....ODD-NUMBERED COLUMN....."
200 FOR PAUSE=1 TO 350:NEXT PAUSE
210 FOR X=137 TO 185 STEP 2
220 FOR Y=66 TO 82
230 PLOT X,Y
240 NEXT Y
250 NEXT X

```


9 GRAPHICS EIGHT

```
260 FOR PAUSE=1 TO 350:NEXT PAUSE
270 ? "J"
280 FOR PAUSE=1 TO 350:NEXT PAUSE
290 ? ".....FOR-NEXT LOOP....."
300 ? ".....COLOR PIXELS IN EVERY....."
310 ? ".....EVEN-NUMBERED COLUMN....."
320 FOR PAUSE=1 TO 350:NEXT PAUSE
330 FOR X=136 TO 186 STEP 2
340 FOR Y=118 TO 132
350 PLOT X,Y
360 NEXT Y
370 NEXT X
380 GOTO 380
RUN
```

The first box drawn is brown. It's that color because *all* pixels—odd *and* even—are illuminated. The second box is blue because the odd-numbered columns are the only ones illuminated, and the third box is green because the even-numbered columns are the only ones illuminated.

Program lines 20 and 30 set the foreground color. Essentially, it takes an odd- and even-numbered pixel to make the color brown in this mode. Remember, the background color influences the foreground color.

Lines 60 to 80, 170 to 190, and 290 to 310 print messages in the TW. (You'll recall from our earlier discussion that the background color controls the TW color.)

Lines 90, 200, and 320 are pause loops, allowing the viewer time to read the message in the TW before each demonstration.

Lines 150 and 270 clear the words from the TW. Nothing else in the program is brand new to you.

An interesting point is made by this demonstration: this mode has such high-resolution, that you have good color even though only half the pixels are illuminated for the second and third boxes.

Play with this program. Remove the pause loops and RUN it. Remove the STEP 2 statements and RUN it. Change the color statements in lines 30 to 50, then RUN it. Change the brightness and RUN it. (Notice that the foreground brightness is a 0 [the darkest], and the background brightness is 12 [nearly the brightest].)

Change the parameters so you have three tall boxes instead of three wide ones. Put the STEP 2 in lines 220 and 340. Be inventive: change, observe, and learn.

The following program introduces the fast and the medium FOR...NEXT loops. It also gives you more practice in artifacting.

Please type in:

```
5 REM LEAVE OUT REMS IF YOU WISH
10 GRAPHICS 8+16:COLOR 1
```



```

15 REM SET BORDER COLOR
20 SETCOLOR 4,6,12
25 REM SET BACKGROUND COLOR
30 SETCOLOR 2,10,12
35 REM SET FOREGROUND COLOR
40 SETCOLOR 1,8,0
45 REM FAST FOR-NEXT LOOP - TOP
50 FOR ROW=30 TO 44
60 PLOT 61,ROW
70 DRAWTO 257,ROW
80 NEXT ROW
85 REM FAST FOR-NEXT LOOP - BOTTOM
87 REM USE THE TRICK
90 FOR ROW=146 TO 160
100 PLOT 61,ROW
110 DRAWTO 257,ROW
120 NEXT ROW
125 REM FAST FOR-NEXT LOOP - UPPER COLUMN
130 FOR COLUMN=155 TO 165 STEP 2
140 PLOT COLUMN,47
150 DRAWTO COLUMN,80
160 NEXT COLUMN
165 REM FAST FOR-NEXT LOOP - LOWER COLUMN
170 FOR COLUMN=155 TO 165 STEP 2
180 PLOT COLUMN,111
190 DRAWTO COLUMN,143
200 NEXT COLUMN
205 REM MEDIUM FOR-NEXT LOOP - MIDDLE BAR
210 FOR COLUMN=0 TO 318 STEP 2
220 PLOT COLUMN,85
230 DRAWTO COLUMN,106
240 NEXT COLUMN
250 GOTO 250
RUN

```

You should see a pink border, an aqua background, two black (or dark brown) rows, two blue columns, and one long green bar in the center.

The fast FOR...NEXT loop is even faster than the XIO fill. I'll explain the slow and the fast FOR...NEXT loops so you'll know why there is a difference.

In both slow and fast ones, there are four parameters:

- 1) the beginning column number
- 2) the ending column number
- 3) the beginning row number
- 4) the ending row number.

In our earlier study of the slow FOR...NEXT loops, we saw that the computer illuminates (PLOTs) each pixel. In the fast ones, the computer illuminates each

9 GRAPHICS EIGHT

pixel but does so by *drawing a line* instead of *PLOTting* each pixel. That's why it's so much faster.

In another chapter, you used pause loops to see the step-by-step process of slow FOR...NEXT loops. Now put in pause loops to slow down the process of fast ones. Please add these to your program. Be sure to use the trick since all the lines are the same, except the line numbers.

```
65 FOR PAUSE=1 TO 150:NEXT PAUSE
75 FOR PAUSE=1 TO 150:NEXT PAUSE
105 FOR PAUSE=1 TO 150:NEXT PAUSE
115 FOR PAUSE=1 TO 150:NEXT PAUSE
145 FOR PAUSE=1 TO 150:NEXT PAUSE
155 FOR PAUSE=1 TO 150:NEXT PAUSE
185 FOR PAUSE=1 TO 150:NEXT PAUSE
195 FOR PAUSE=1 TO 150:NEXT PAUSE
225 FOR PAUSE=1 TO 150:NEXT PAUSE
235 FOR PAUSE=1 TO 150:NEXT PAUSE
RUN
```

Now you can see the computer **PLOT** a dot at 61,30 (line 60), then **DRAW** a line **TO** 257,30 (line 70). The instruction that follows tells the computer to do the next row, meaning the computer will place the cursor at 61,31, put a dot, then **DRAW** a line **TO** 257,31.

In essence, this fast FOR...NEXT loop will **DRAW** fifteen lines (rows 30 to 44) from column 61 **TO** column 257. Looking at it another way, your computer has two instructions to follow a total of fifteen times: it must **PLOT** at one point and **DRAWTO** another point. It does this fifteen times because there are fifteen lines to be drawn in rows 30 to 44.

In the slow FOR...NEXT loop, you recall, the computer **PLOTS** each pixel. In this instance, the computer would have to **PLOT** each pixel in columns 61 through 257 (197 columns), and in each row from 30 to 44 (fifteen rows) for a total of 2,955 pixels (197×15). It is considerably slower to illuminate 2,955 pixels individually than to draw fifteen lines.

Notice that the four FOR...NEXT loops at the beginning have one thing in common: they all use the two shortest parameters. For example, lines 50 through 80 use rows 20 to 44 for the FOR...NEXT loop, *not* column 61 to column 257. Lines 170 through 200 use column 155 to column 165 for the FOR...NEXT loop, *not* row 111 to row 143. When you use the shortest expanse of numbers in the FOR...NEXT loop, fewer lines are drawn. The first four boxes are fast FOR...NEXT loops.

Lines 210 through 240 reverse the trend. The FOR...NEXT loop contains the *greatest* expanse of numbers: 0 to 318. This is the medium FOR...NEXT loop. Use this approach here because you must illuminate the even-numbered pixels only in all the columns, and you do that with the STEP 2 statement. If you had used the

row numbers to make this a fast FOR...NEXT loop, you would not get the third color. When you skip *columns*, you get another color; when you skip *rows*, you do not.

Even though this is the medium speed FOR...NEXT loop, it is still much faster than the "normal" FOR...NEXT loop, and faster than the XIO fill.

Now is the time for a suggestion: design a program that shows you understand how to use color in this mode.

Have the program compare the "regular" PLOTting FOR...NEXT loop, the XIO fill, the medium FOR...NEXT loop, and the fast FOR...NEXT loop. Make each box the same size so you can compare the speed of the different approaches. You'll be quite impressed with the fast FOR...NEXT loop, which is at least *ten times* faster than the PLOTting FOR...NEXT loop. You may even want to time the drawing of each box. *Remember*, your parameters are 0 to 319, and 0 to 191 in GR.8+16.

What follows is a picture in GR.8 that gives the illusion of depth. It's a long program to type in, but I think it will be worth it, especially if you follow my final suggestions about changing it. The picture shows a highway in the middle of nowhere. Don't forget to use the trick.

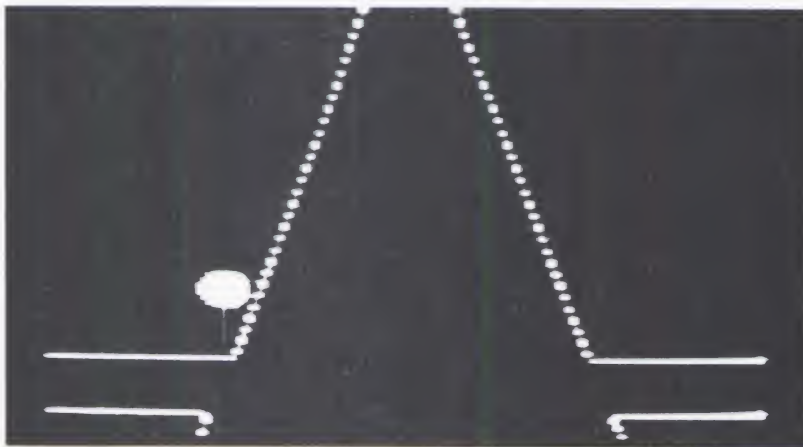
```

10 GRAPHICS 8+16:COLOR 1
20 SETCOLOR 1,12,14:SETCOLOR 2,0,0
30 PLOT 80,159:DRAWTO 140,21:REM LEFT ROAD
40 PLOT 240,159:DRAWTO 180,21:REM RIGHT ROAD
50 PLOT 160,159:DRAWTO 160,149:REM FIRST LINE
60 PLOT 160,139:DRAWTO 160,130:REM SECOND LINE
70 PLOT 160,121:DRAWTO 160,113:REM THIRD
80 PLOT 160,106:DRAWTO 160,98:REM FOURTH
90 PLOT 160,91:DRAWTO 160,85:REM FIFTH
100 PLOT 160,79:DRAWTO 160,74:REM SIXTH
110 PLOT 160,70:DRAWTO 160,66:REM SEVENTH
120 PLOT 160,61:DRAWTO 160,57:REM EIGHTH
130 PLOT 160,52:DRAWTO 160,49:REM NINTH
140 PLOT 160,45:DRAWTO 160,42:REM TENTH
150 PLOT 160,38:DRAWTO 160,36:REM ELEVENTH
160 PLOT 160,33:DRAWTO 160,32:REM TWELFTH
170 PLOT 160,30:DRAWTO 160,28:PLOT 160,26
180 PLOT 160,24:PLOT 160,22
190 REM SKY
200 FOR ROW=0 TO 20
210 PLOT 0,ROW
220 DRAWTO 319,ROW
230 NEXT ROW
240 PLOT 79,159:DRAWTO 0,159:REM LEFT UPPER ROAD
250 PLOT 241,159:DRAWTO 319,159:REM RIGHT
    UPPER ROAD

```


9 GRAPHICS EIGHT

```
260 PLOT 0,182:DRAWTO 66,182:REM LOWER LEFT ROAD
270 PLOT 250,182:DRAWTO 319,182:REM LOWER
    RIGHT ROAD
280 PLOT 68,182:DRAWTO 65,191:REM LEFT
    SHORT ROAD
290 PLOT 250,183:DRAWTO 255,191:REM RIGHT
    SHORT ROAD
300 PLOT 76,153:DRAWTO 76,138:REM STOP SIGN BASE
310 PLOT 71,137:DRAWTO 81,137:REM STOP SIGN
    LOWER BAR
320 PLOT 83,136:DRAWTO 86,133:REM LOWER RIGHT
    OF STOP SIGN
330 PLOT 86,133:DRAWTO 86,129:REM RIGHT SIDE
    OF STOP SIGN
340 DRAWTO 82,125:REM UPPER RIGHT SIDE OF
    STOP SIGN
350 DRAWTO 71,125:REM TOP OF STOP SIGN
360 DRAWTO 67,129:REM UPPER LEFT
370 DRAWTO 67,133:REM LEFT SIDE
380 DRAWTO 71,137
390 REM FILL-IN STOP SIGN
400 PLOT 70,136:DRAWTO 82,136
410 PLOT 69,135:DRAWTO 83,135
420 PLOT 68,134:DRAWTO 84,134
430 PLOT 68,133:DRAWTO 85,133
440 PLOT 67,132:DRAWTO 86,132
450 PLOT 67,131:DRAWTO 86,131
460 PLOT 67,130:DRAWTO 86,130
470 PLOT 67,129:DRAWTO 86,129
480 PLOT 68,128:DRAWTO 85,128
490 PLOT 69,127:DRAWTO 84,127
500 PLOT 70,126:DRAWTO 83,126
510 GOTO 510
RUN
```



Consider changing the picture by adding a house, a cow, or a car. Now there's a challenge!

Before leaving this mode, try to use three or four fast FOR...NEXT loops with varied STEP statements. Make one FOR...NEXT loop a STEP 3, another a STEP 6, and so forth. You can draw some very arresting displays this way and make an interesting background and/or introduction for your programs. Also, try the above with the same parameters, but with different STEP statements.

NOTE TO NON-GTIA AND NON-XL OWNERS

The succeeding chapters concern graphics modes that are not directly addressable on your model Atari. As is the case throughout the book, new techniques will be introduced in many of the programs, and I urge you to learn more about your computer by studying them. With this in mind, I offer two suggestions.

The first one is to read *all* the following chapters. You won't be able to type in the programs as they are, but you'll pick up information from the general and specific discussions about individual programs, even though you can't see them in action.

The second suggestion is that you re-write the programs in a graphics mode that you do have. You'll have to alter the column and row parameters and change the color approach, but you can approximate the programs.

Some programs take advantage of specific characteristics of the new graphics modes. You won't be able to duplicate them perfectly, but you can give it a whirl. You have some hard work ahead of you, but it could prove most rewarding. You'll have to use paper (graph paper is best) to draw the pictures. Though you need the XL model to RUN programs in GR.14 and 15, you can LIST them on the 400 and 800 machines after you receive the "ERROR" message.

In addition, when there is a new technique or approach that I think you should see in a program, I'll add a program at the end of the chapter just for you. My explanation of it won't be as complete as the one that appears earlier in the chapter (that would be too repetitious). Make sure you read the whole chapter, for it will help you understand your separate program. Wherever it proves to be helpful—and not too repetitive—I'll discuss your program.

Be sure to glance at the end of a chapter to see if there is indeed a special program for you. If so, you won't need to re-write the earlier one. Enjoy the challenge—and good luck!

NOTE TO GTIA AND XL OWNERS

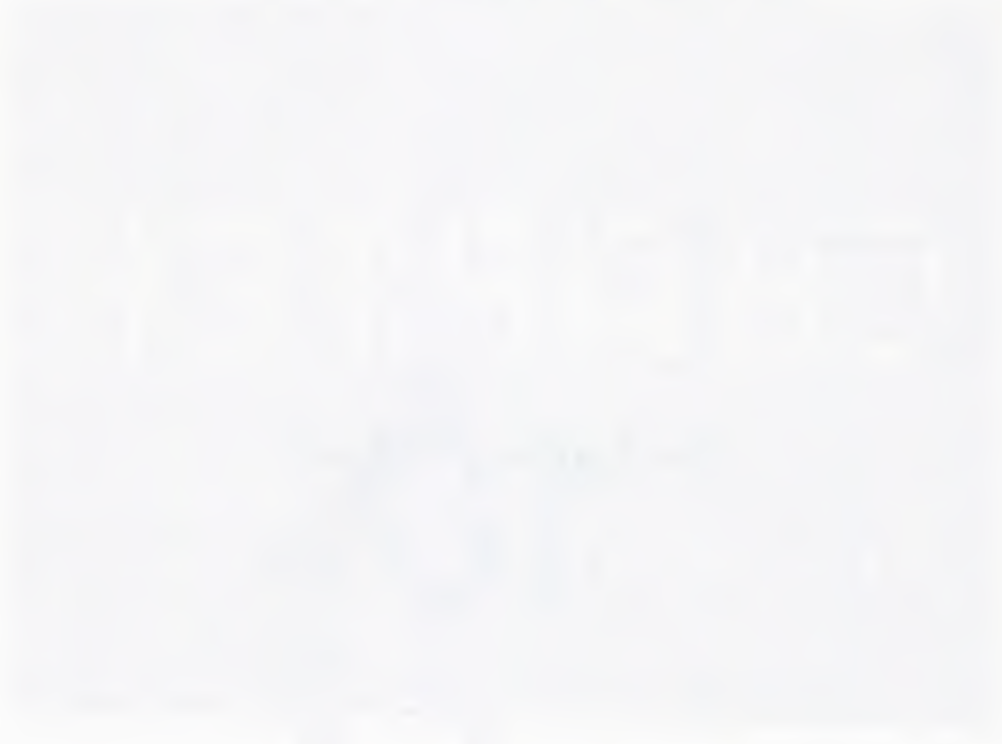
Some of you do not have an XL model, while others don't have the GTIA chip. Consequently, some of you cannot directly address GRAPHICS 9, 10, 11, 14, and 15.

Since this is the case, I'll occasionally put in a program at the end of a chapter that's akin to the one (or ones) you typed in. Though similar, it will not be identical, so I suggest you also type it in. The special program at the end may introduce a new way of doing something, in order to approximate an effect you saw in programs in GR.9, 10, 11, 14, and 15.

GRAPHICS NINE



THE ADIUTANT



Graphics Nine

Graphics modes 9, 10, and 11 are directly addressable if you have an XL machine or if you have a computer with the new GTIA chip. If you bought your Atari 400 or 800 after 1981, chances are that you have this newer computer chip. If you're not sure you have the GTIA chip, type this in:

```
10 GRAPHICS 9
20 GOTO 20
RUN
```

If you have the chip, your screen will turn the background color *black*.

If your screen does not change, you have the CTIA chip that does not include three new modes. For under \$100, you can have the GTIA chip installed and you'll be able to take advantage of these new graphics modes for your programming.

Before we get into the specifics of GRAPHICS 9, let's discuss variables. Then I'll show you a new way to use them that drastically reduces the length of many of your programs.

You've used variables throughout this book. When you type `FOR PAUSE=1 TO 200:NEXT PAUSE`, the *variable* is the word "PAUSE." It is called a variable because the value of it can vary. In this case, the value changes from 1 to 200 in increments of one. First, the value of PAUSE is 1, then 2, then 3, and so on, until it reaches 200.

Let's look at another way you have used variables:

```
FOR X=10 TO 20
```

10 GRAPHICS NINE

```
FOR Y=80 TO 100
PL.X,Y
NEXT Y
NEXT X
```

This slow FOR...NEXT loop PLOTs every pixel in columns 10 through 20, and PLOTs every pixel in rows 80 through 100. You have two variables (X and Y), and their values change by increments of 1. First, your computer illuminates the pixel at 10,80, then at 10,81, at 10,82, etc. Eventually it PLOTs at 11,80, then at 11,81, and so on, until it has illuminated (or PLOTted) all the columns and rows. It can accomplish this because the values of the variables change.

You have seen yet another way to change the value of the variable:

```
FOR X=1 TO 33 STEP 2
```

When you type this in, the computer knows the value of the variable will be 1, then 3, then 5, etc., until the value is 33.

PAUSE, X, and Y are all variables, and your computer changes their values according to your instructions. Now you'll learn another way to use variables and change their values. Type in this program, please:

```
10 X=10
20 PRINT X
30 X=X+9
40 GOTO 20
RUN
```

To stop this endless loop, press the BREAK key. In line 10, you set the value of the variable X at 10. This is called initializing the variable, meaning you give the variable its initial (or beginning) value.

Because of line 20, your computer prints that value (10). In line 30, you tell the computer that X is no longer 10; it is now $10 + 9$, or 19. In line 40, your computer is sent to line 20, setting up the endless loop. In line 20, you tell the computer to "PRINT X," but this time it will print 19, the *new* value of X (because of what you've set up in line 30). It will then go to line 30 again, change the value of X to 28, go on to line 40 where it will be sent to line 20, and so forth.

In this program, your computer will print a string of numbers that starts with 10 and increases by 9 each time it prints again (10, 19, 28, 37, etc.). At this point, let's discuss the specifics of GRAPHICS 9, then we'll use this new approach to change the value of a variable.

Modes 9, 10, and 11 all have 80 columns and 192 rows. As in all the other modes, the numbering starts at 0, so the columns are numbered from 0 to 79, and the rows are numbered from 0 to 191. There is no TW in these modes.

In GR.9, you can use only one color, but you get sixteen shades of that color. There are two ways to put color on the screen; let's use the easiest method first.

Type in this program. It will show you how to use color in this mode—the easy way—then we'll go on to the “changing variable” technique.

The SETCOLOR 4,2,0 Technique for Color

```
10 GRAPHICS 9 (you don't need the +16 because there is no TW)
20 SETCOLOR 4,2,0
30 COLOR 12
40 PLOT 0,0
50 DRAWTO 79,191
90 GOTO 90
RUN
```

In this method of placing color, first you use SETCOLOR 4; second, you choose a color from 0 to 15; and third, you put it in. Finally, you put in a 0. In this case, we chose 2 for the color:

```
SETCOLOR 4,2,0
```

The number you place after the 4 (0 to 15), chooses the basic color. The sixteen shades of that one color are now available for your drawings.

You chose the shade of that color by putting in the statement COLOR 12 in line 30.

Run your cursor up to line 30. Change it to COLOR 9 and RUN it. Now you see another shade of color number 2. To repeat, the *second* number in the SETCOLOR statement chooses the basic color.

The number after the COLOR statement chooses the shade. It's a bit confusing, but you'll get the hang of it. Each time you want to draw in a different shade, you put in another COLOR statement. Add the following to your program, please:

```
60 COLOR 14
70 PLOT 0,191
80 DRAWTO 79,0
RUN
```

You have sixteen shades you can use, so just add another number after the COLOR statement. Unusual things happen, however, when you use a brightness other than 0 in the SETCOLOR statement. I'll leave this for you to experiment with because so many things do happen. Try writing a program using FOR...NEXT loops to show the different brightnesses with the different color combinations.

One last thing. When you use SETCOLOR 4,COLOR,0, you can't use COLOR 0 which is used for the background and border color. Why don't you change the two COLOR statements to 1 and 15? Then you'll see the darkest and brightest shades of color number 2 (SETCOLOR 4,2,0 line 20). If you like, change

10 GRAPHICS NINE

the number following SETCOLOR 4 and see the darkest and brightest shades of that color.

Now that you know the basics of color in GR.9, let's go on to the "changing variable" technique and see how helpful it is. We'll take this new approach slowly. In the following program, we'll draw a few lines with a fast FOR...NEXT loop. Then we'll change the SHADE of the color by changing the value of the variable shade.

Type in the following program, please. (The REMs are helpers, but you don't have to type them in.)

```
10 GRAPHICS 9
20 SHADE=1:REM INITIALIZE VALUE OF THE VARIABLE
   NAMED SHADE
30 FOR REPEAT=1 TO 15:REM DO LINES 30 TO 120
   FIFTEEN TIMES
40 SETCOLOR 4,8,0:REM CHOOSE COLOR EIGHT
50 COLOR SHADE:REM THE FIRST SHADE OF COLOR
   EIGHT. IN THIS CASE, IT IS SHADE ONE
60 FOR X=36 TO 44:REM COLUMNS
70 PLOT X,0:REM PL.36,0 THE FIRST TIME THROUGH
   THE LOOP
80 DRAWTO X,191:REM DRAWTO 36,191 THE FIRST TIME
   THROUGH THE LOOP
90 NEXT X:REM GO TO COLUMN 37 AND DRAW A LINE TO
   ROW 191. KEEP DRAWING LINES IN COLUMNS
   38,39,ETC TO ROW 191
100 FOR PAUSE=1 TO 200:NEXT PAUSE:REM WAIT LOOP
110 SHADE=SHADE+1:REM CHANGE THE VALUE OF THE
   VARIABLE SHADE BY ADDING A ONE TO IT.
120 NEXT REPEAT:REM RETURN TO BEGINNING OF
   REPEAT LOOP(LINE 30). GO THRU SAME PROCESS
   15 TIMES MORE.
130 GOTO 130:REM DO NOT REVERT(DEFAULT) TO
   GRAPHICS ZERO
RUN
```

There are two keys to this program. The first is the REPEAT loop, lines 30 through 120. Everything between these line numbers will be done a total of fifteen times.

The second key is line 110. Each time the computer reaches this line number, it changes the value of the variable SHADE so that the lines will change and display all shades.

The lines will first be drawn in SHADE 1 (lines 20 and 40); secondly, the number of the shade changes to 2 because of line 110 (the lines will then be drawn in SHADE 2). Each time through the REPEAT loop, the value of SHADE will be increased by one.

Let's look at a program that's a little more complicated. This time we'll change the value of three variables: the column number, the row number, and the SHADE number. Type this in, please:

```

10 GRAPHICS 9:X=10:Y=0:SHADE=15
20 FOR REPEAT=1 TO 15
30 SETCOLOR 4,12,0:COLOR SHADE
40 PLOT X,Y
50 DRAWTO X,191
60 X=X+4:Y=Y+10:SHADE=SHADE-1
70 NEXT REPEAT
80 GOTO 80
RUN

```

What you have here is a simple program that goes through a complicated routine because of the changing variable.

By the time your computer reaches line 60, it has drawn a line from column 10, row 0 to column 10, row 191 in SHADE 15 (the brightest) of COLOR 12.

When it reaches line 60, these are the variables and their values:

```

X= 10
Y=  0
SHADE= 15

```

Because of line 60, however, the values are changed; X becomes 14; Y becomes 10; and SHADE becomes 14.

In line 70, the computer is sent back to line 20. Your Atari will now draw a line from column 14, row 10 to column 14, row 191 in SHADE 14 (a lighter shade of COLOR 12). When it reaches line 60 again, the values of the variables are changed until:

```

X= 18
Y= 20
SHADE= 13

```

The computer will do this process fifteen times, drawing lines that are four columns apart, ten rows shorter, and one shade lighter until you have fifteen lines that have decreased in length and brightness.

Think of the program you'd have to write if you didn't use the changing variable technique. In previous programs, you've done approximately the same thing, but the programs were much longer.

Why not make some changes in this program? Change the value of SHADE in line 10 to 1, then change the value of SHADE in line 60 to +1 and RUN it. You'll now get the lines drawn from darkest to brightest.

Experiment with the other variables and see what you come up with. Recall,

though, that the columns go from 0 to 79, and the rows go from 0 to 191. You must stay inside those parameters.

The POKE 712, COLOR * 16+0 Technique for Color

Before we leave the program, let's look at the second way to use color in this mode. Make this change in your program, please:

```
30 POKE 712,12*16+0: COLOR SHADE
RUN
```

You have the same fifteen colors again. Your computer stores things in memory locations. In this graphics mode, there are two ways to store the color information.

The first way is to use the SETCOLOR 4,COLOR,0 approach. The second is to use POKE 712, COLOR * 16+0. If you wish, you can do the multiplication yourself in the latter method: POKE 712, 128+0. This is the same as POKE 712,8*16+0. (You can also do this for your graphics mode choice GR.19 instead of GR.3+16.)

Let's use a brightness other than 0 and see what happens:

```
20 POKE 712,2*16+6: COLOR SHADE
RUN
```

When you use a brightness other than 0, you lose shades in either method.

REVIEW

There are two methods for putting color on the screen in GR.9: the first is SETCOLOR 4,COLOR,0; the second is POKE 712,COLOR*16+0. Using either method, the color information is stored in a memory location in your computer. If you wish to speed up the latter method, do the multiplication yourself, which means multiply the color number by 16.

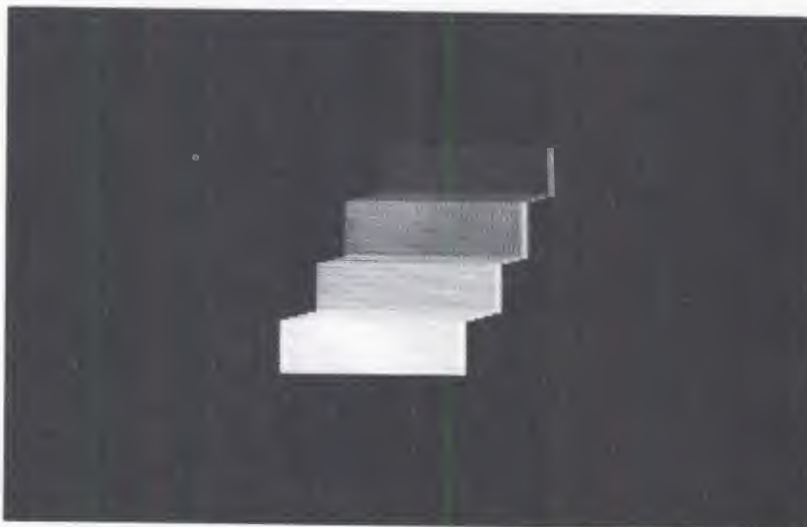
Remember, if you add a brightness value in either of these methods, you will *not* get your sixteen shades. You may use a specific color with just a few shades. In this case, use a brightness number other than 0 and experiment.

One more thing you can do in GR.9 is draw something that appears to have depth as a result of the different shades. The darker the shade, the "deeper" the object appears.

Look at the following program. It draws a picture of a flight of stairs. The different shades cause a 3D effect. Notice how you put in a different color number to change the shade of the drawing.

This is a practice program. Type it in just to see the demonstration. I recommend, however, that you type it in, see what happens, then re-write the program

using fast FOR...NEXT loops and, if possible, the changing variable technique. If you can do both things, you're on your way to mastering the two techniques.



```

10 GRAPHICS 9:SETCOLOR 4,0,0
20 COLOR 15
30 FOR X=30 TO 50
40 FOR Y=86 TO 106
50 PLOT X,Y
60 NEXT Y
70 NEXT X
100 COLOR 14
110 PLOT 31,85:DRAWTO 51,85
120 PLOT 32,84:DRAWTO 52,84
130 PLOT 33,83:DRAWTO 53,83
140 PLOT 34,82:DRAWTO 54,82
200 COLOR 13
210 FOR X=34 TO 54
220 FOR Y=64 TO 81
230 PLOT X,Y
240 NEXT Y
250 NEXT X
300 COLOR 12
310 PLOT 35,63:DRAWTO 55,63
320 PLOT 36,62:DRAWTO 56,62
330 PLOT 37,61:DRAWTO 57,61
400 COLOR 11
410 FOR X=37 TO 57
420 FOR Y=40 TO 60
430 PLOT X,Y
440 NEXT Y
450 NEXT X

```

10 GRAPHICS NINE

```
500 COLOR 10
510 PLOT 38,39:DRAWTO 58,39
520 PLOT 39,38:DRAWTO 59,38
530 PLOT 40,37:DRAWTO 60,37
600 COLOR 9
610 FOR X=40 TO 60
620 FOR Y=19 TO 36
630 PLOT X,Y
640 NEXT Y
650 NEXT X
700 COLOR 8
710 PLOT 41,18:DRAWTO 61,18
720 PLOT 42,17:DRAWTO 62,17
800 GOTO 800
RUN
```

Make corrections or whatever changes you desire.

Program for Non-GTIA and Non-XL Owners

The following program demonstrates the same changing variable technique. It draws eight lines which decrease in length. We'll use the eight brightnesses here since we can't use the sixteen shades. The effect, therefore, will be different. Type this in, please:

```
10 GRAPHICS 5+16:X=78:Y=3:BRITE=14
20 SETCOLOR 4,10,0
30 FOR REPEAT=1 TO 8
40 COLOR 1:SETCOLOR 0,4,BRITE
50 PLOT X,Y
60 DRAWTO X,47
70 X=X-7:Y=Y+4:BRITE=BRITE-2
80 FOR PAUSE=1 TO 200:NEXT PAUSE
90 NEXT REPEAT
100 GOTO 100
RUN
```

Your computer draws a line in COLOR 4, BRIGHTNESS 14 from column 78, row 3, to column 78, row 47. This takes us through line 60.

In line 70, the values of the variables are changed. X becomes 71; Y becomes 7; and BRITE becomes 12.

Line 80 contains a pause loop. Line 90 sends the computer to line 30. In line 40, the variable BRITE is now 12.

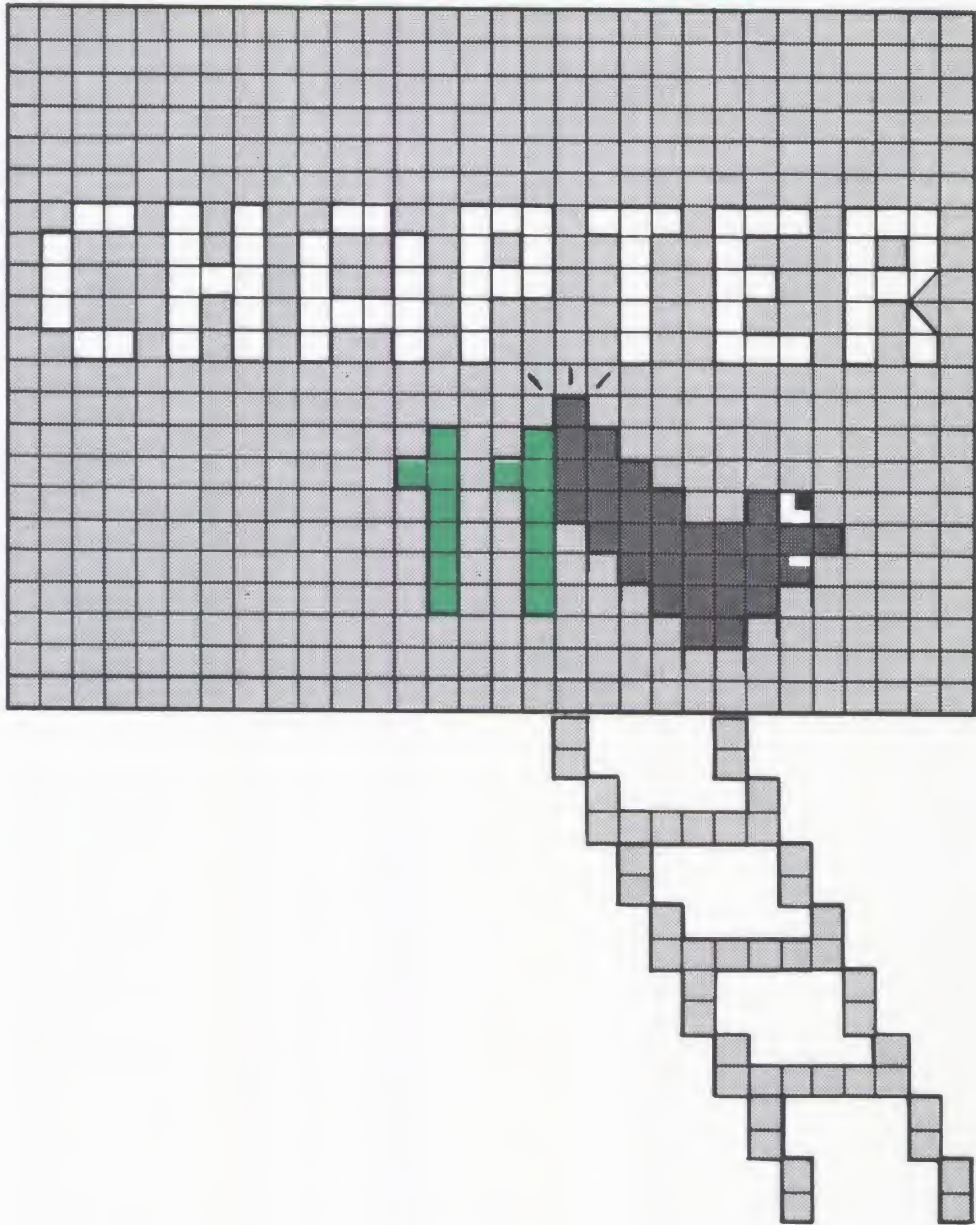
In lines 50 and 60, the computer will PLOT and DRAWTO according to the changed values of the variables, and the Atari will draw a line from column 71, row 7 to column 71, row 47. A switch occurs here from the earlier program in that the brightness changes. Accordingly, the brightness of the first and second lines changes to BRIGHTNESS 12.

In line 70, the variables are changed again and a new line will be drawn from a different column and row number in a different brightness.

If you're a trifle confused, re-read the entire chapter. Also, experiment with pause loops so you can observe these changes at a slower speed.

If you're feeling ambitious, eliminate the SETCOLOR statement and use three or four COLORS for the changing variables instead of brightnesses. Or, you may use the IF...AND...THEN approach to changing the colors. Let's look at GRAPHICS 10 now.

GRAPHICS TEN



THE UNIVERSITY OF CHICAGO



Graphics Ten

In GRAPHICS 10, you can put nine separate colors on the screen simultaneously. This is accomplished by using the four color registers normally reserved for player/missile graphics. These four registers can only be used with POKEs. Their memory locations, or addresses are 704, 705, 706, and 707. To use a color with a POKE, you have to multiply the color number by 16 (you saw this earlier), then add the brightness, if there is one. For example, if you want COLOR 4 and BRIGHTNESS 6 in address 706, multiply the color number by 16 (4×16) then add the brightness, which is 6. You end up with POKE 706, $4 \times 16 + 6$, and you can let the computer compute for you, or you can do the multiplication yourself and type POKE 706,70 ($4 \times 16 + 6$).

The POKEs for color are from 704 through 712. As mentioned earlier, POKEs 704 through 707 are normally used for P/M graphics. In this mode, however, POKE 704 controls the background *and* border color. You then have one background/border color that you choose with POKE 704, and eight foreground (PLOTting) colors for your pictures (POKEs 705-712). The next program produces eight columns in different colors. It will also show you how to use the GOSUB and RETURN statements. Type in the following program lines please:

```
10 GRAPHICS 10:POKE 704,9*16+2:COLOR 0
20 POKE 705,1*16+2:COLOR 1
30 PLOT 9,191:DRAWTO 9,0:DRAWTO 2,0:POSITION
   2,191:POKE 765,1
40 GOSUB 900
100 POKE 706,3*16+4:COLOR 2
```

11 GRAPHICS TEN

```
110 PLOT 19,191:DRAWTO 19,0:DRAWTO 12,0:POSITION
    12,191:POKE 765,2
120 GOSUB 900
200 POKE 707,5*16+6:COLOR 3
210 PLOT 29,191:DRAWTO 29,0:DRAWTO 22,0:POSITION
    22,191:POKE 765,3
220 GOSUB 900
300 POKE 708,7*16+12:COLOR 4
310 PLOT 39,191:DRAWTO 39,0:DRAWTO 32,0:POSITION
    32,191:POKE 765,4
320 GOSUB 900
400 POKE 709,8*16+6:COLOR 5
410 PLOT 49,191:DRAWTO 49,0:DRAWTO 42,0:POSITION
    42,191:POKE 765,5
420 GOSUB 900
500 POKE 710,11*16+8:COLOR 6
510 PLOT 59,191:DRAWTO 59,0:DRAWTO 52,0:POSITION
    52,191:POKE 765,6
520 GOSUB 900
600 POKE 711,13*16+2:COLOR 7
610 PLOT 69,191:DRAWTO 69,0:DRAWTO 62,0:POSITION
    62,191:POKE 765,7
620 GOSUB 900
700 POKE 712,15*16+4:COLOR 8
710 PLOT 79,191:DRAWTO 79,0:DRAWTO 72,0:POSITION
    72,191:POKE 765,8
720 GOSUB 900
800 GOTO 800
900 XIO 18,#6,0,0,"S":RETURN
RUN
```

You see eight variously colored columns with a different border/background color, for a total of nine colors.

Look at line 700. There is a 15 for the color number after the POKE:

POKE 712,15*16+4

There is, also, a color statement at the end of the line:

:COLOR 8

Now look at the end of line 710. The number used for the XIO fill is the one that follows the COLOR statement:

:POKE 765,8

A specific POKE represents a specific COLOR number. The following chart may make all this a little easier to understand.

POKE 704 is for COLOR 0 (line 10 - POKE 704.... : COLOR 0)

POKE 705 is for COLOR 1 (line 20 - POKE 705.... : COLOR 1)

POKE 706 is for COLOR 2 (line 100 - POKE 706.... : COLOR 2)
 POKE 707 is for COLOR 3 (line 200 - POKE 707.... : COLOR 3)
 POKE 708 is for COLOR 4 (line 300 - POKE 708.... : COLOR 4)
 POKE 709 is for COLOR 5 (line 400 - POKE 709.... : COLOR 5)
 POKE 710 is for COLOR 6 (line 500 - POKE 710.... : COLOR 6)
 POKE 711 is for COLOR 7 (line 600 - POKE 711.... : COLOR 7)
 POKE 712 is for COLOR 8 (line 700 - POKE 712.... : COLOR 8)

You can see, for another example, that POKE 706 in line 100 uses COLOR 2, regardless of the number after POKE 706. That's why you have to use a 2 after the POKE 765 for the XIO fill: POKE 765,2 (line 110).

Notice the COLOR 1 through COLOR 8 statements. These color numbers are the ones to use with the XIO fill statements because the colors correspond with POKE 705 through POKE 712 for eight colors.

In line 30, you put in the PLOTs, DRAWTOs, and POKEd the correct number into POKE 765. The only thing left to do is put in the XIO fill. Rather than type the fill eight times, we put it at the *end* of the program. Each time it's needed, we tell the computer to GOSUB (GO to the SUBroutine at) line 900, use the XIO fill, then RETURN to the instructions following the GOSUB statement. (If you had GOSUB 900: PLOT 30,30 in line 40, the computer would RETURN and PLOT a dot of color at 30,30). Using the GOSUB/RETURN allows us to type the XIO 18,#6,0,0,"S:" only once instead of eight times.

I used the XIO fill in this program because the POKE 765 works so unusually. Now that you know how to use the XIO, however, you can change the program.

You can substitute fast FOR...NEXT loops, slow FOR...NEXT loops, or put in changing variables. Implementing the latter is easy because you'd have to change only the X coordinates. I'll leave this in your capable hands.

Before we leave this program, look at the GOTO statement at line 800. Change it to line 1000, see what happens, and figure out why it does. Remember, the computer carries out your instructions according to the line numbers.

There is another way to place color in GR.10. For the last five colors, you can use SETCOLOR statements.

USE

SETCOLOR 0,COLOR,BRIGHTNESS
 SETCOLOR 1,COLOR,BRIGHTNESS
 SETCOLOR 2,COLOR,BRIGHTNESS
 SETCOLOR 3,COLOR,BRIGHTNESS
 SETCOLOR 4,COLOR,BRIGHTNESS

INSTEAD OF

POKE 708,COLOR*16,+BRIGHTNESS
 POKE 709,COLOR*16,+BRIGHTNESS
 POKE 710,COLOR*16,+BRIGHTNESS
 POKE 711,COLOR*16,+BRIGHTNESS
 POKE 712,COLOR*16,+BRIGHTNESS

When I use GR.10, I use the POKEs. The SETCOLOR approach only complicates matters.

11 GRAPHICS TEN

In case you're interested, the following program demonstrates how to use the SETCOLOR method. It produces staggered lines of different colors. Type this in, please:

```
10 GRAPHICS 10
20 POKE 705,120:COLOR 1
30 PLOT 4,10:DRAWTO 9,191
40 POKE 706,2*16+12:COLOR 2
50 PLOT 14,10:DRAWTO 19,191
60 POKE 707,5*16+2:COLOR 3
70 PLOT 24,10:DRAWTO 29,191
80 SETCOLOR 0,15,4:COLOR 4
90 PLOT 34,10:DRAWTO 39,191
100 SETCOLOR 1,11,4:COLOR 5
110 PLOT 44,10:DRAWTO 49,191
120 SETCOLOR 2,5,8:COLOR 6
130 PLOT 54,10:DRAWTO 59,191
140 SETCOLOR 3,6,12:COLOR 7
150 PLOT 64,10:DRAWTO 69,191
160 SETCOLOR 4,9,12:COLOR 8
170 PLOT 74,10:DRAWTO 79,191
180 GOTO 180
RUN
```



The staggered lines create a nice effect. Note that this is accomplished by PLOTting in one column, then DRAWing TO a column five numbers to the right, a perfect set-up for a fast FOR...NEXT loop and/or a changing variable.

Notice also that lines 80, 100, 120, 140, and 160 use SETCOLOR statements, not POKEs. The SETCOLOR statements take the place of the POKEs (708 through 712). Look at lines 20 and 40 to see the two different ways you can put color in a POKE.

One final suggestion about this program will increase your know-how. Notice that we did not change the background color. If you wanted to, you could use `POKE 704,COLOR times 16 plus BRIGHTNESS`. Make this change by putting the `POKE` statement in line 10.

Before we progress to the next program, you should learn a peculiarity of the `FOR...NEXT` loop. It will be much easier to explain what happens as you watch it on the screen, so please type in the following:

```

10 FOR X=1 TO 10
20 PRINT X
30 NEXT X
40 PRINT
50 PRINT
60 PRINT "AFTER THE FINAL 'NEXT X', THE VALUE
   IS ONE NUMBER HIGHER THAN IT 'SHOULD' BE
70 PRINT X
80 GOTO 80
RUN

```

You should have:

```

1
2
3
4
5
6
7
8
9
10

```

AFTER THE FINAL 'NEXT X,' THE VALUE IS ONE NUMBER HIGHER THAN IT 'SHOULD' BE.

```

11

```

The value 1 is assigned in the `FOR` statement in line 10. After the 1 is printed (line 20), the computer goes on to line 30 for the `NEXT X` statement. At this point, a number is added to the 1. Since there is no `STEP` statement in line 10, the value added is 1. The computer returns to line 10 where the value is now $1 + 1$, or 2. It then prints 2 (line 20).

In line 30, a 1 is added again: $2 + 1$. The computer returns to line 10 where the value is now 3. Line 20 prints the number 3.

When the computer reaches line 30, it reads `NEXT X` and adds 1 to the number. Eventually, that makes `X` equal 11, so the computer does not return to line 10. Instead, it continues on, skips two lines (because of lines 40 and 50), and prints the final value of `X`. In this case, `X` equals 11.

11 GRAPHICS TEN

You will see the results of all this in our demonstration program. Knowing how the FOR...NEXT loop works is important in your programming. Sometimes, you need to reset the value of your variables, otherwise the final NEXT X gives you a value you don't want or expect later in your program. Because of this peculiarity, I had to reset the value of a variable in line 90 in the next program. (We'll discuss it when we go through the program.)

If you find my explanation of "the quirk" satisfactory, please skip the following program. But for those wishing to delve deeper into this FOR...NEXT loop quirk, I present a program that may help you better visualize what is happening. (I won't explain the techniques I've used now because I describe them later on in the book.)

```
10 FOR X=1 TO 3:PRINT "LINE 10. X=";X
20 FOR PAUSE=1 TO 1000:NEXT PAUSE
30 PRINT "LINE 30. X=";X:PRINT X
40 PRINT
50 FOR PAUSE=1 TO 1000:NEXT PAUSE
60 PRINT "LINE 60. X=";X:NEXT X:PRINT "AFTER THE
   FINAL 'NEXT X', X=";X:PRINT X
70 PRINT
80 PRINT
90 FOR PAUSE=1 TO 1000:NEXT PAUSE
100 PRINT "LINE 100. X=";X
110 GOTO 110
RUN
```

The next program uses six variables that change. Additionally, there are four lines that decrease in length; each uses a different POKE for color. Then there are four lines that increase in length; once again, each uses a different POKE for color.

Eight lines (of changing lengths) in eight different colors will be displayed. You can do this complicated task in only sixteen program lines because the values of the variables change in lines 70 and 140.

Type this in, please:

```
10 GRAPHICS 10:X=20:Y=26:NBR=705:KLR=2:BRT=0:K=1
20 POKE 704,8*16+0:REM BACKGROUND COLOR
30 FOR REPEAT=1 TO 4
40 POKE NBR,KLR*16+BRT:COLOR K
50 PLOT X,Y
60 DRAWTO 60,Y
70 X=X+2:Y=Y+20:NBR=NBR+1:KLR=KLR+2:BRT=BRT+2:
   K=K+1
80 NEXT REPEAT
90 X=26
100 FOR AGAIN=5 TO 8
110 POKE NBR,KLR*16+BRT:COLOR K
120 PLOT X,Y
```

```

130 DRAWTO 60,Y
140 X=X-2:Y=Y+20:NBR=NBR+1:KLR=KLR-2:BRT=BRT+2:
    K=K+1
150 NEXT AGAIN
160 GOTO 160
RUN

```

The program would be several pages long if it weren't for the changing variable technique coupled with FOR...NEXT loops. It's important to add this technique to your repertoire because it's a real work horse.

Due to the six changing variables, half of this book would be required to take you step-by-step through this entire program. What we'll do then, is look at **one variable** all the way through. Afterward, I recommend that you take another variable (or variables) and walk it (or them) through.

Line 10	—	the variable X is initialized at a value of 20.
Line 50	—	the computer PLOTs at 20,Y.
Line 60	—	the computer DRAWs a line TO 60,Y
Line 70	—	the value of X is changed to 22.
Line 80	—	the computer returns to line 30.
Line 50	—	the computer PLOTs at 22,Y.
Line 60	—	the computer DRAWS a line TO 60,Y
Line 70	—	the value of X is changed to 24.
Line 80	—	the computer returns TO line 30.
Line 50	—	the computer PLOTs at 24,Y.
Line 60	—	the computer DRAWs a line TO 60,Y
Line 70	—	the value of X is changed to 26.
Line 80	—	the computer returns to line 30.
Line 50	—	the computer PLOTs at 26,Y.
Line 60	—	the computer DRAWs a line TO 60,Y
Line 70	—	the value of X is changed to 28.
Line 80	—	the computer will not return to line 30 because it has been through the repeat loop four times already.

This is "the quirk" you learned about earlier. The value is 28, not 26. We have to make allowances for the quirk if we don't want X to equal 28.

The computer goes on to line 90. Here, the value of X is changed from 28 to 26 so that the two middle lines have the same column number, 26.

When they go through the REPEAT loop, each of the six variables change in line 70. The numbers that are added may differ, but the process is the same. Since the variables are not all reset in line 90, however, the "quirk numbers" are the ones

11 GRAPHICS TEN

used at the beginning of the next FOR...NEXT loop (line 100) for the variables Y, NBR, KLR, BRT, and K. (If you have the idea, you can skip the next part.) We continue to line 100, with X equalling 26. Another FOR...NEXT loop begins.

Line 120 — the computer PLOTs at 26,Y.
Line 130 — the computer DRAWs a line TO 60,Y
Line 140 — the value of X is changed to 24($X=X-2$)
Line 150 — go back to line 100.
Line 120 — the computer plots at 24,Y.
Line 130 — the computer DRAWs a line TO 60,Y
Line 140 — the value of X is changed to 22.
Line 150 — go back to line 100.
Line 120 — the computer plots at 22,Y.
Line 130 — the computer DRAWs a line TO 60,Y
Line 140 — the value of X is changed to 20.
Line 150 — go back to line 100.
Line 120 — the computer plots at 20,Y.
Line 130 — the computer DRAWs a line TO 60,Y
Line 140 — the value of X is changed to 18.
Line 150 — the computer will not go back to line 100
because it has been through the loop four
times already.

If you put in a PRINT X statement at line 155, you'd see the value of X is 18, although the number 18 is never used. 18 is the "quirk number." If this were a longer program that used X again, you'd see the importance of knowing about this quirk in the FOR...NEXT loops. Thus concludes our run-through of the program.

If you're tempted to give up at this point, *please don't!* The program may look overwhelming, but it isn't if you break it down as I did. For example, look at the variable NBR. You can see in line 30 that NBR stands for the POKE NUMBER. Since NBR equals 705 (line 10), you know that line 40 translates to POKE 705. Follow this variable all the way through, and you'll see that the number increases by 1 each time through every loop. If you do this, remembering the "quirk number" will be used in line 100 (since it wasn't reset in line 90), you'll do fine. Recall, too, that the "quirk numbers" are used for *all* variables, except X.

If you still are confused, re-write the entire program the long way (in other words, do not use variables or FOR...NEXT loops). Line 40, for example, becomes:

40 POKE 705, 2*16+0: COLOR 1

Continue in this fashion, translating *all* the variables to numbers, then eliminate line 70 and do the math yourself, putting in the new numbers.

Your program will end up considerably longer than this one—but length is unimportant here and understanding is the goal.

Program For Non-GTIA and Non-XL Owners

The next program is a bit different. The changing variables control the length of the lines, the color of the background, and the brightness of the drawing color. Additionally, one of the variables is changed *twice*. Type this in, please:

```

10 GR.5+16:X=24:Y=23:K=14:B=0
20 COLOR 1
30 FOR OUTLP=1 TO 4
40 SETCOLOR 4,K,0:SETCOLOR 0,6,B
50 FOR INLP=1 TO 3
60 PL.X,Y
70 DR.X,47
80 X=X+1
90 FOR J=1 TO 200:NEXT J
100 NEXT INLP
110 X=X+7:K=K-3:B=B+4
120 FOR J=1 TO 200:NEXT J
125 NEXT OUTLP
130 GOTO 130
RUN

```

To understand the program, you may have to re-read the explanation of the earlier one, substituting the values from this program.

One of the controllers here is line 110. Each time the computer reaches this line, it will add 7 to the value of X, subtract 3 from the value of the COLOR, and add 4 to the BRIGHTNESS of the drawing color.

One of the major differences between this program and the earlier one, however, is the FOR...NEXT INLP. This inloop causes the computer to draw three lines instead of one because of line 80.

When the computer reaches line 80 the first time, the value of X is 24 (line 10). At this point, the computer adds 1 to the value of X making it 25. It then returns to line 50 and goes through the loop again, drawing a line in column 25. It does so again, drawing a line in column 26.

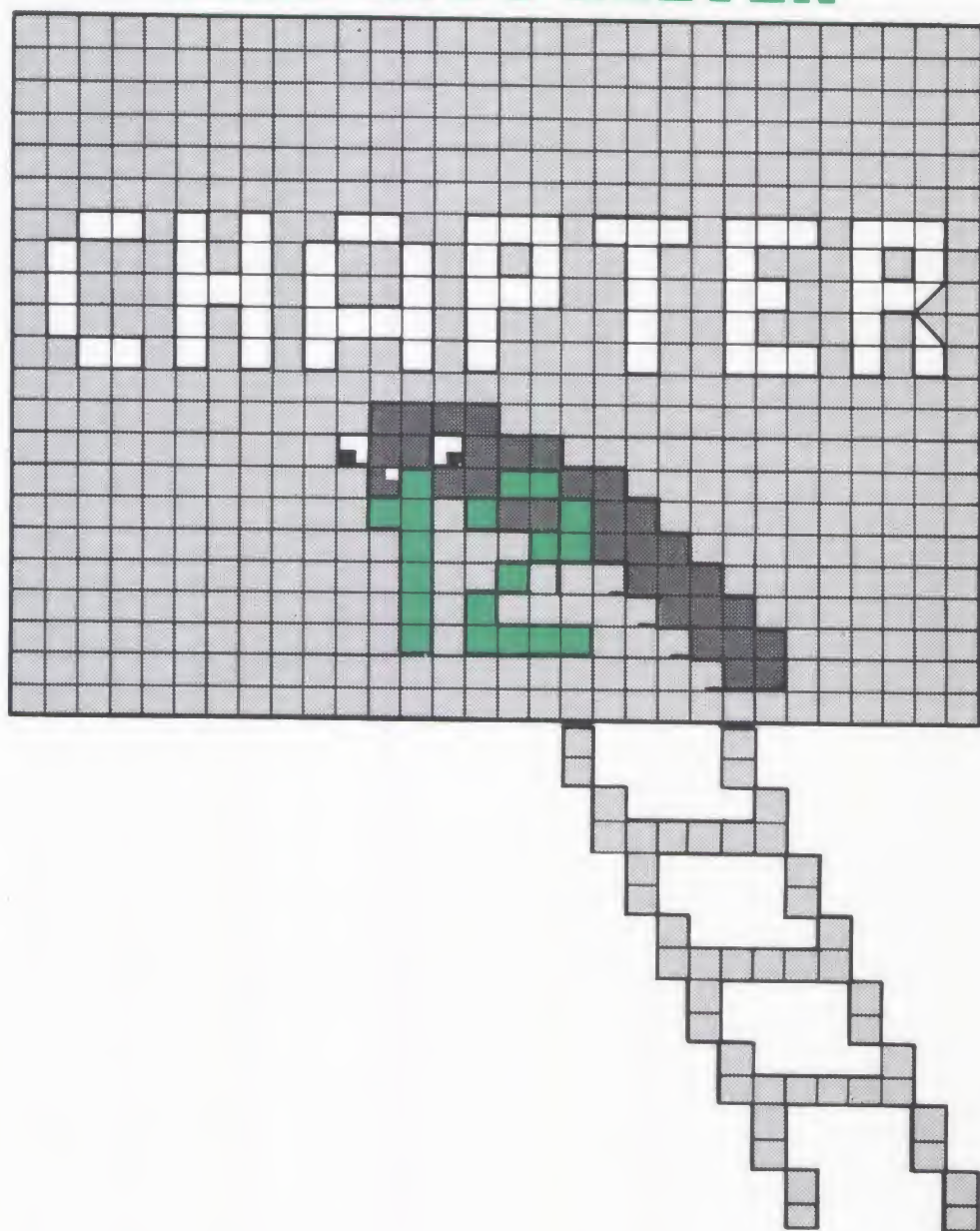
Because of the quirk, however, the value of X is 27. Since we want to draw lines at columns 24, 34, 44, and 54, we had to add 7, *not* 8, as one might assume if one didn't know about the quirk.

The computer will draw three lines a total of four times. They will begin in columns 24, 34, 44, and 54 because we have taken into consideration the quirk factor.

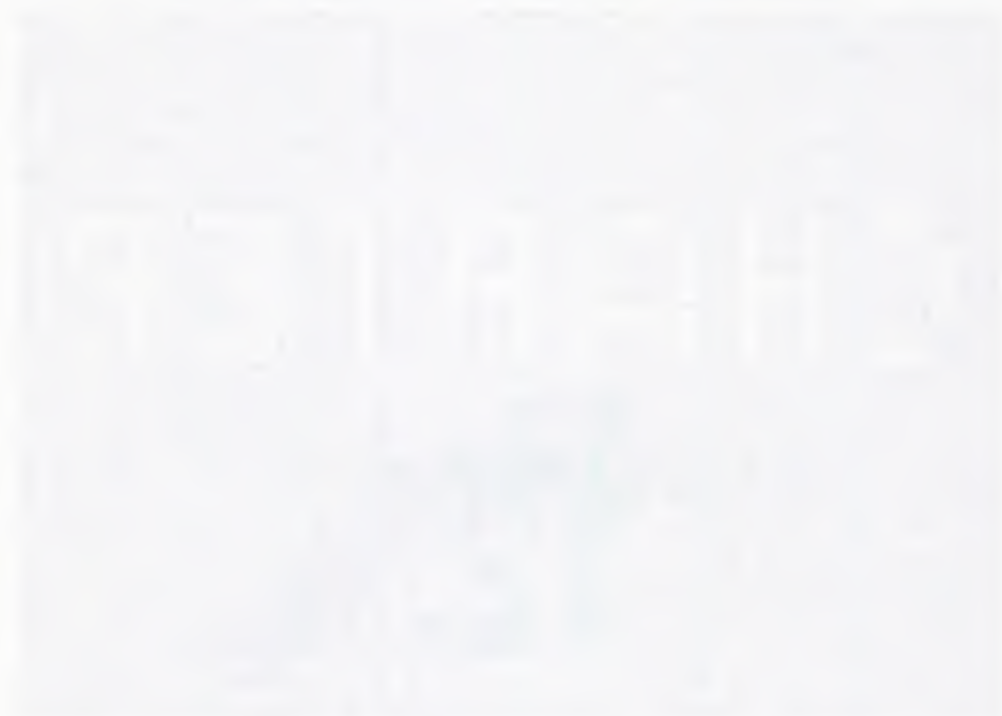
11 GRAPHICS TEN

If you're still a tiny bit confused, experiment by putting messages in a TW, explaining what's happening. You could refer to the first program back on page 140 to see how this is done.

GRAPHICS ELEVEN



RECEIVED



Graphics Eleven

GRAPHICS 11 is the last of the directly-accessible GTIA modes that we'll study. Like the others, it has 80 columns and 192 rows. In this mode, you can display fifteen foreground colors, but you can use only *one* BRIGHTNESS for all fifteen colors.

To begin the process of using color, first choose the BRIGHTNESS. You can use two methods:

```
POKE 712,BRIGHTNESS (0 TO 14, even numbers only)
```

or

```
SETCOLOR 4,0,BRIGHTNESS (0 TO 14, even numbers only)
```

After choosing one of the above methods, use a simple COLOR statement (1 to 15) to obtain a drawing color.

The border/background is set to COLOR 0, black, giving you a total of sixteen colors for your drawing. There is *no* TW in this mode.

For clarity's sake, I'm including an easy program that shows you how to use color in GR.11.

Please type in:

```
10 GRAPHICS 11
20 POKE 712,8:REM CHOOSE BRIGHTNESS FOR 15
   FOREGROUND COLORS
30 COLOR 1:REM CHOOSE ONE FOREGROUND COLOR
```

12 GRAPHICS ELEVEN

```
40 PLOT 5,30
50 DRAWTO 25,90
60 COLOR 5
70 PLOT 20,30
80 DRAWTO 40,90
90 COLOR 10
100 PLOT 35,30
110 DRAWTO 55,90
120 COLOR 15
130 PLOT 50,30
140 DRAWTO 70,90
150 GOTO 150
RUN
```

You wind up with four bent lines, each in a different color. Use your cursor to change the color numbers in lines 30, 60, 90, and 120. Make them COLOR 0, 4, 9, and 14, and RUN it. You now have three lines. COLOR 0 is used for the black background, so you may *not* use it for a foreground color.

You could continue changing the COLOR statements until you have seen all fifteen foreground colors, but that's unnecessary. The important thing is understanding the process of putting color on the screen. First choose a BRIGHTNESS, then use a COLOR statement (1 to 15).

Move your cursor to line 20, please. Change the 8 to a 14. Change the color statement in line 30 to COLOR 2 and RUN it. Now you see the brightest level for the four colors.

Next, move the cursor to line 20, change the 14 to a 0 and RUN it. You now see the darkest level for these four colors.

You can do much with this program to practice the techniques you've learned. You know how to:

- 1) cycle the brightnesses
- 2) cycle the colors
- 3) set up an infinite loop with color changes
- 4) remove the PLOT and DRAWTO statements and replace them with XIO fill statements
- 5) draw all lines in one color, then STEP through the colors, or
- 6) make a FOR...NEXT loop and use the changing variable approach.

I've put in one easy variation of the program to arouse your interest. Make these changes, please:

```
10 GRAPHICS 11:M=1
30 COLOR M
60 COLOR M
90 COLOR M
120 COLOR M
```



```
150 M=M+1
160 IF M>15 THEN M=1
170 GOTO 30
RUN
```

The program will continue cycling the colors due to lines 150 and 170. Use your BREAK key to stop it.

The math signs to use in line 160 are:

=	equal to
>	greater than
<	less than
=>	equal to or greater than
<=	less than or equal to

Line 160 says, "If M is greater than 15, then M=1."

This next program places all fifteen colors on the screen at once. The blending of the colors at the bottom is very colorful. Type this in, please:

```
10 GRAPHICS 11:X=9:K=1
20 POKE 712,10
30 FOR REPEAT=1 TO 15
40 COLOR K
50 PLOT X,10:DRAWTO 40,191
60 X=X+4:K=K+1
70 NEXT REPEAT
80 GOTO 80
RUN
```



You have an attractive effect because all lines are drawn to column 40, row 191 (line 50). Of course, the key is *line 60* for the changing variables.

Before we get into our next program, I'll take a moment to digress. When an artist paints a picture, he usually doesn't want anyone else to see it until it's finished. The colors and brushes he chooses, his techniques, the changes he makes, *indeed* the entire process of creation is important to the artist—but not necessarily to his audience. The finished work of art is the goal.

As a computer artist, you can do the same thing in many graphics modes (at the end of this chapter, I'll show you how). But, because of the medium in which you work, you can let anyone see the step-by-step creation of your picture. If you wish, you can involve the viewer in your art from the very first line you draw or the first dot of color you place on your "canvas." There is an immediacy to this approach that captures the imagination of the viewer: he is *there* as you create. It is this aspect of your creation that places a burden on you with which the painter is unconcerned. Whether it takes him three years or thirty to produce his piece of art, no one knows or cares. This is not the case for the computer artist; the viewer *does* care how long it takes to place something on the screen.

Thus, you—or I—have to choose. Do I want to place the completed picture on the screen as soon as possible, or is it important to involve the viewer, step-by-step?

I wouldn't want to play Pac-Man if I had to wait each time for the screen slowly to be drawn. The picture is not the thing here, but the game is.

Conversely, there are times I like to see a picture as it's being drawn, and not when it's all finished. The following program falls into the former category.

My choice may be a bad one; some of yours may be, too. This is for the individual viewer to decide, but at least we *have* a choice.

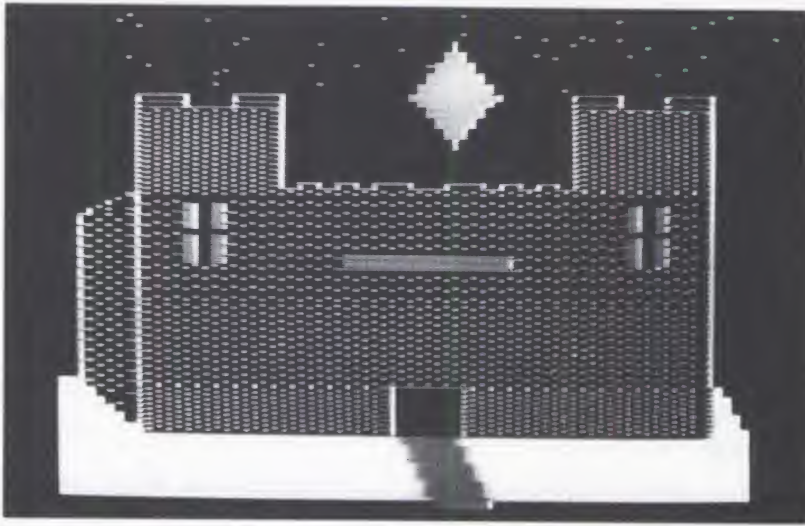
Can you imagine the state of art today if every artist did the same thing the same way? A ludicrous ideal! Don't, therefore, do things exactly the same way everyone else does them. Make unique decisions and try an approach or a statement that is truly yours.

There are many choices to be made. The first is *how* to present the picture. Make that choice, then apply your talents and skills to produce various effects. Now for the program.

My son, Darren, who is eleven, loves castles and was the inspiration for the following one. We'll do this picture a bit differently, since it's so long. Every so often, I'll ask you to RUN the program. In this way, you can see what you've done so far. We'll discuss whatever's significant in the portion you've typed in.

Please enter the following lines, using the trick as often as possible:

```
10 GRAPHICS 11
20 POKE 712,4:REM CHOOSE BRIGHTNESS OF
  FOREGROUND COLORS
```

```

30 COLOR 13
40 PLOT 70,160:DRAWTO 70,30:DRAWTO 65,30
50 DRAWTO 65,35:DRAWTO 60,35:DRAWTO 60,30
60 DRAWTO 55,30:DRAWTO 55,65:DRAWTO 53,65
70 DRAWTO 53,63:DRAWTO 51,63:DRAWTO 51,65
80 DRAWTO 49,65:DRAWTO 49,63:DRAWTO 47,63
90 DRAWTO 47,65:DRAWTO 45,65:DRAWTO 45,63
100 DRAWTO 41,63:DRAWTO 41,65:DRAWTO 37,65
110 DRAWTO 37,63:DRAWTO 33,63:DRAWTO 33,65
120 DRAWTO 31,65:DRAWTO 31,63:DRAWTO 29,63
130 DRAWTO 29,65:DRAWTO 27,65:DRAWTO 27,63
140 DRAWTO 25,63:DRAWTO 25,65:DRAWTO 23,65
150 DRAWTO 23,30:DRAWTO 18,30:DRAWTO 18,35
160 DRAWTO 13,35:DRAWTO 13,30:DRAWTO 8,30
170 DRAWTO 8,160:DRAWTO 35,160:DRAWTO 35,140
180 DRAWTO 43,140:DRAWTO 43,160:DRAWTO 70,160
190 FOR PAUSE=1 TO 100:NEXT PAUSE
200 GOTO 200
RUN

```

You have drawn the outline of the castle. If something looks wrong, check your program with the one in the book. Remove line 200 and continue typing, please.

```

200 COLOR 1
210 FOR X=9 TO 33 STEP 2
220 FOR Y=158 TO 142 STEP -2
230 PLOT X,Y:NEXT Y:NEXT X
240 FOR X=8 TO 34 STEP 2
250 FOR Y=157 TO 141 STEP -2
260 PLOT X,Y:NEXT Y:NEXT X
270 FOR PAUSE=1 TO 100:NEXT PAUSE

```


12 GRAPHICS ELEVEN

```
280 GOTO  
RUN
```

As you see, this draws the first layer of bricks on the left side. Please delete line 280 and continue typing.

```
280 FOR X=44 TO 70 STEP 2  
290 FOR Y=157 TO 141 STEP -2  
300 PLOT X,Y:NEXT Y:NEXT X  
310 FOR X=43 TO 69 STEP 2  
320 FOR Y=158 TO 142 STEP -2  
330 PLOT X,Y:NEXT Y:NEXT X  
340 FOR PAUSE=1 TO 100:NEXT PAUSE  
350 GOTO 350  
RUN
```

You've drawn the first layer of bricks on the right side. Please delete line 350 and continue typing.

```
350 FOR X=8 TO 68 STEP 2  
360 FOR Y=140 TO 108 STEP -3  
370 PLOT X,Y:NEXT Y:NEXT X  
380 FOR X=9 TO 69 STEP 2  
390 FOR Y=139 TO 109 STEP -3  
400 PLOT X,Y:NEXT Y:NEXT X  
410 FOR PAUSE=1 TO 100:NEXT PAUSE  
420 GOTO 420  
RUN
```

The second layer of bricks is now done. Please delete line 420 and continue typing.

```
420 FOR X=8 TO 68 STEP 2  
430 FOR Y=106 TO 66 STEP -4  
440 PLOT X,Y:NEXT Y:NEXT X  
450 FOR X=9 TO 69 STEP 2  
460 FOR Y=107 TO 67 STEP -4  
470 PLOT X,Y:NEXT Y:NEXT X  
480 FOR PAUSE=1 TO 100:NEXT PAUSE  
490 GOTO 490  
RUN
```

Now you've got the third layer of bricks. Please delete line 490 and continue typing.

```
490 FOR X=8 TO 22 STEP 2  
500 FOR Y=64 TO 36 STEP -2  
510 PLOT X,Y:NEXT Y:NEXT X  
520 FOR X=9 TO 21 STEP 2  
530 FOR Y=65 TO 37 STEP -2
```

```
540 PLOT X,Y:NEXT Y:NEXT X
550 FOR PAUSE=1 TO 100:NEXT PAUSE
560 GOTO 560
RUN
```

This draws the fourth layer of bricks, left side. Please delete line 560 and continue typing.

```
560 PLOT 8,35:DRAWTO 23,35
570 PLOT 8,33:DRAWTO 13,33
580 PLOT 18,33:DRAWTO 23,33
590 FOR PAUSE=1 TO 100:NEXT PAUSE
600 GOTO 600
RUN
```

This draws the large bricks on the top, left. Please delete line 600 and continue typing.

```
600 FOR X=55 TO 69 STEP 2
610 FOR Y=64 TO 36 STEP -2
620 PLOT X,Y:NEXT Y:NEXT X
630 FOR X=56 TO 70 STEP 2
640 FOR Y=65 TO 37 STEP -2
650 PLOT X,Y:NEXT Y:NEXT X
660 FOR PAUSE=1 TO 100:NEXT PAUSE
670 GOTO 670
RUN
```

The fourth row of bricks, right side, is now done. Please delete line 670 and continue typing.

```
670 PLOT 55,35:DRAWTO 70,35
680 PLOT 55,33:DRAWTO 60,33
690 PLOT 65,33:DRAWTO 70,33
700 FOR PAUSE=1 TO 100:NEXT PAUSE
710 GOTO 710
RUN
```

This draws the large bricks on the top, right. Please delete line 710 and continue typing.

```
710 COLOR 0
720 FOR X=30 TO 48
730 FOR Y=90 TO 95
740 PLOT X,Y:NEXT Y:NEXT X
750 FOR PAUSE=1 TO 100:NEXT PAUSE
760 GOTO 760
RUN
```

Lines 420 through 470 drew the third layer of bricks. Now you have just erased a portion of that layer so you can draw something else in there in a different color.

12 GRAPHICS ELEVEN

(It is more difficult to create the third layer if you have to keep a portion of it free of bricks.)

Please delete line 760 and continue typing.

```
760 COLOR 1
770 POKE 765,1
780 PLOT 48,95:DRAWTO 48,90
790 DRAWTO 30,90:POSITION 30,95
800 XIO 18,#6,0,0,"D:"
810 FOR PAUSE=1 TO 100:NEXT PAUSE
820 GOTO 820
RUN
```

This draws the light coming through the long window in the middle. It takes the place of the black background you drew in lines 710 through 740. Delete line 820 and continue typing, please.

```
820 COLOR 0
830 FOR X=13 TO 17
840 FOR Y=70 TO 93
850 PLOT X,Y:NEXT Y:NEXT X
860 FOR PAUSE=1 TO 100:NEXT PAUSE
870 GOTO 870
RUN
```

You have just erased a section of bricks in the upper left-hand portion of the castle. Please delete line 870 and continue typing.

```
870 COLOR 1
880 POKE 765,1
890 PLOT 17,93:DRAWTO 17,71
900 DRAWTO 13,71:POSITION 13,93
910 XIO 18,#6,0,0,"S:"
920 FOR PAUSE=1 TO 100:NEXT PAUSE
930 GOTO 930
RUN
```

This uses the XIO fill to put light in the upper left-hand corner window. Please delete line 930 and continue typing.

```
930 COLOR 0
940 FOR X=61 TO 65
950 FOR Y=70 TO 93
960 PLOT X,Y:NEXT Y:NEXT X
970 FOR PAUSE=1 TO 100:NEXT PAUSE
980 GOTO 980
RUN
```

The bricks in the upper right-hand window have been erased. You are now ready to put in the light. Please delete line 980 and continue typing.


```

980 COLOR 1:POKE 765,1
990 PLOT 65,93:DRAWTO 65,71
1000 DRAWTO 61,71:POSITION 61,93
1010 XIO 18,#6,0,0,"S:"
1020 FOR PAUSE=1 TO 100:NEXT PAUSE
1030 GOTO 1030
RUN

```

This produces the light shining through the window in the upper right-hand portion of the castle. Please delete line 1030 and continue typing.

```

1030 COLOR 0
1040 PLOT 15,70:DRAWTO 15,93
1050 PLOT 13,81:DRAWTO 17,81
1060 PLOT 13,82:DRAWTO 17,82
1070 FOR PAUSE=1 TO 100:NEXT PAUSE
1080 GOTO 1080
RUN

```

The bars in the upper left-hand window have been drawn. Notice lines 1050 and 1060. We had to use a double-thickness horizontal line because the vertical ones are so much fatter. Delete line 1080 and continue typing, please.

```

1080 PLOT 63,70:DRAWTO 63,93
1090 PLOT 61,81:DRAWTO 65,81
1100 PLOT 61,82:DRAWTO 65,82
1110 FOR PAUSE=1 TO 100:NEXT PAUSE
1120 GOTO 1120
RUN

```

You used the same COLOR 0 (line 1030) to draw the bars in the upper right-hand window. Again, you used doubly-thick horizontal lines. Please delete line 1120 and continue typing.

```

1120 COLOR 9
1130 POKE 765,9
1140 PLOT 74,185:DRAWTO 74,161
1150 DRAWTO 4,161:POSITION 4,185
1160 XIO 18,#6,0,0,"S:"
1170 FOR PAUSE=1 TO 100:NEXT PAUSE
1180 GOTO 1180
RUN

```

You've drawn the water in front of the castle. Please delete line 1180 and continue typing.

```

1180 PLOT 74,160:DRAWTO 71,140
1190 PLOT 73,160:DRAWTO 71,141
1200 PLOT 72,160:DRAWTO 71,142
1210 PLOT 71,160:DRAWTO 71,143
1220 FOR PAUSE=1 TO 100:NEXT PAUSE

```

12 GRAPHICS ELEVEN

```
1230 GOTO 1230
RUN
```

This uses the same COLOR 9 from line 1120 to draw water to the right of the castle. Please delete line 1230 and continue typing.

```
1230 COLOR 13
1240 PLOT 8,160:DRAWTO 2,140
1250 DRAWTO 2,75:DRAWTO 8,66
1260 FOR X=2 TO 6 STEP 2
1270 FOR Y=141 TO 75 STEP -4
1280 PLOT X,Y:NEXT Y:NEXT X
1290 FOR X=3 TO 7 STEP 2
1300 FOR Y=139 TO 75 STEP -4
1310 PLOT X,Y:NEXT Y:NEXT X
1320 PLOT 5,73:PLOT 7,73
1330 PLOT 6,148:PLOT 7,73
1340 PLOT 7,144:PLOT 5,143
1350 FOR PAUSE=1 TO 100:NEXT PAUSE
1360 GOTO 1360
RUN
```

You've drawn the left side of the castle. Showing one side like this adds the element of depth. It's as if you're looking at the castle from the left side of it. Please delete line 1360 and continue typing.

```
1360 COLOR 9
1370 FOR X=0 TO 3
1380 FOR Y=161 TO 185
1390 PLOT X,Y:NEXT Y:NEXT X
1400 PLOT 0,160:DRAWTO 0,138
1410 PLOT 1,160:DRAWTO 1,138
1420 PLOT 2,160:DRAWTO 2,141
1430 PLOT 3,160:DRAWTO 3,143
1440 PLOT 4,160:DRAWTO 4,149
1450 PLOT 5,160:DRAWTO 5,152
1460 PLOT 6,160:DRAWTO 6,156
1470 PLOT 7,157:DRAWTO 7,160
1480 FOR PAUSE=1 TO 100:NEXT PAUSE
1490 GOTO 1490
RUN
```

This draws the water to the left of the castle. Please delete line 1490 and continue typing.

```
1490 COLOR 0
1500 PLOT 46,188:DRAWTO 42,160
1510 PLOT 45,188:DRAWTO 41,160
1520 PLOT 44,188:DRAWTO 40,160
1530 PLOT 43,188:DRAWTO 39,160
```

```

1540 PLOT 42,188:DRAWTO 38,160
1550 PLOT 41,188:DRAWTO 37,160
1560 PLOT 40,188:DRAWTO 36,160
1570 FOR PAUSE=1 TO 100:NEXT PAUSE
1580 GOTO 1580
RUN

```

You've erased the water so you can put in the drawbridge. Please delete line 1580 and continue typing.

```

1580 COLOR 14
1590 PLOT 46,188:DRAWTO 42,160
1600 PLOT 45,188:DRAWTO 41,160
1610 PLOT 44,188:DRAWTO 40,160
1620 PLOT 43,188:DRAWTO 39,160
1630 PLOT 42,188:DRAWTO 38,160
1640 PLOT 41,188:DRAWTO 37,160
1650 PLOT 40,188:DRAWTO 36,160
1660 FOR PAUSE=1 TO 100:NEXT PAUSE
1670 GOTO 1670
RUN

```

This drew the drawbridge. No matter what you choose here—XIO fill or otherwise—you'll also have some PLOTting and DRAWing TO to do. Please delete line 1670 and continue typing (you're almost finished).

```

1670 COLOR 3
1680 PLOT 42,10:DRAWTO 42,50
1690 PLOT 43,14:DRAWTO 43,46
1700 PLOT 44,18:DRAWTO 44,42
1710 PLOT 45,22:DRAWTO 45,38
1720 PLOT 46,26:DRAWTO 46,34
1730 PLOT 47,30:DRAWTO 47,32
1740 PLOT 41,14:DRAWTO 41,46
1750 PLOT 40,18:DRAWTO 40,42
1760 PLOT 39,22:DRAWTO 39,38
1770 PLOT 38,26:DRAWTO 38,34
1780 PLOT 37,30:DRAWTO 37,32
1790 FOR PAUSE=1 TO 100:NEXT PAUSE
1800 GOTO 1800
RUN

```

Now you've drawn the moon. Please delete 1800 and continue typing.

```

1800 COLOR 3
1810 FOR STAR=1 TO 43
1820 X=INT(80*RND(1))
1830 Y=INT(29*RND(1))
1840 PLOT X,Y
1850 NEXT STAR
1860 GOTO 1860
RUN

```


12 GRAPHICS ELEVEN

This last part sprinkles stars above the castle. It tells the computer to put approximately forty-three stars at RaNDom (RND) in coordinates 0 to 79 (X), and coordinates 0 to 28 (Y). This sets up a rectangle within which the stars are randomly placed. You have approximately forty-three stars because the computer may place one star on top of another, once or several times. Consider using this approach in the highway picture in GRAPHICS 8 to give the illusion of a star-filled horizon.

Program For Non-GTIA and Non-XL Owners

Earlier, I mentioned invisible drawing. First, let's look at a picture drawn visibly, then change it a bit and draw invisibly. Type this in, please:

```
10 GRAPHICS 3+16
20 COLOR 1:SETCOLOR 0,4,6
30 FOR X=0 TO 39 STEP 2
40 PLOT X,0:DRAWTO X,23
50 NEXT X
60 COLOR 2:SETCOLOR 1,8,12
70 FOR Y=1 TO 23 STEP 3
80 PLOT 0,Y:DRAWTO 39,Y
90 NEXT Y
120 GOTO 120
RUN
```

You have criss-crossing bars drawn visibly. Now make the following changes, and you'll draw invisibly.

```
20 COLOR 1:SETCOLOR 0,0,0
60 COLOR 2:SETCOLOR 1,0,0
100 SETCOLOR 0,8,14
110 SETCOLOR 1,7,6
RUN
```

This time the picture was drawn, but you didn't see it until it was completely finished. You can do this because of the SETCOLOR system on your Atari.

You put the color black into SETCOLOR 0 in line 20. In line 60, you changed the color information in SETCOLOR 0, so the black color was replaced with a pale blue color.

Recall that when you change the color information in a SETCOLOR statement, anything you've previously drawn is instantly changed to the new color. You used the same process in SETCOLOR 1 in lines 60 and 110. That's all there is to drawing invisibly!

GRAPHICS FOURTEEN



Graphics Fourteen

GR.14	—	160 X 160
GR.14+16	—	160 X 192

This is a one-color, two-luminance mode. Use COLOR 1 for drawing. Change the color by using SETCOLOR 0,COLOR,BRIGHTNESS. For the border/background, use SETCOLOR 4,COLOR,BRIGHTNESS. Use SETCOLOR 2,COLOR,BRIGHTNESS for the text window.

In this chapter you'll learn two more ways to change variables. Let's look at one of these methods, which I'll call the "X + or -, Y + or -" method.

The X + or -, Y + or - Method

With this method, you initialize the variables. Then you change them by added to or subtracting from them, as in the changing variable technique—but it's done differently. Type in this program, please, and you'll see the difference.

```

10 GRAPHICS 14+16:COLOR 1:X=70:Y=76
20 SETCOLOR 0,1,14
30 PLOT X,Y:REM (70,76)
50 DRAWTO X+20,Y:REM (90,76)
70 DRAWTO X+20,Y+10:REM (90,86)
90 DRAWTO X,Y+10:REM (70,86)
110 DRAWTO X,Y+20:REM (70,96)
130 DRAWTO X+20,Y+20:REM (90,96)
150 DRAWTO X+20,Y+30:REM (90,106)
170 DRAWTO X,Y+30:REM (70,106)

```

13 GRAPHICS FOURTEEN

```
180 GOTO 180  
RUN
```

This program draws something similar to a “backward E.” The PLOT and DRAWTO coordinates in the parentheses clarify what is happening here. The method isn’t difficult, so there’s no need for a long explanation. Employ it when the changing variable technique would be awkward or impossible. In the future, you’ll find a use for this method that may not be apparent right now. Furthermore, other programmers use it, so you’ll need to know what they’re doing.

Let’s change the BRIGHTNESS of the lines to dress up this program. We’ll use the technique you learned about at the end of the previous chapter.

Add these lines, please:

```
40 SETCOLOR 0,1,12  
60 SETCOLOR 0,1,10  
80 SETCOLOR 0,1,8  
100 SETCOLOR 0,1,6  
120 SETCOLOR 0,1,4  
140 SETCOLOR 0,1,2  
160 SETCOLOR 0,1,0  
RUN
```

The first line is drawn in BRIGHTNESS 14, the second is drawn in BRIGHTNESS 12 and the first line is also changed to that BRIGHTNESS. The third line is drawn in BRIGHTNESS 10, and the first and second lines are changed to that BRIGHTNESS, too. The process continues.

If you wish, use changing variables instead. Also, you can put in some pause loops after each SETCOLOR or DRAWTO statement; you could change the COLOR number instead of the BRIGHTNESS in the SETCOLOR statement, too.

Let’s go on to the next method which I’ll dub the “variable number of variables” method.

The Variable Number of Variables Method

So far for drawings, you’ve used just two variables: X and Y. You *don’t have to* call them X and Y. Computer users do because X is considered to be the column coordinate, and Y is considered to be the row coordinate. Furthermore, you don’t have to use just two variables (in the next program, you use four). What you can do with four variables is not easily done with two when employing the changing variable technique.

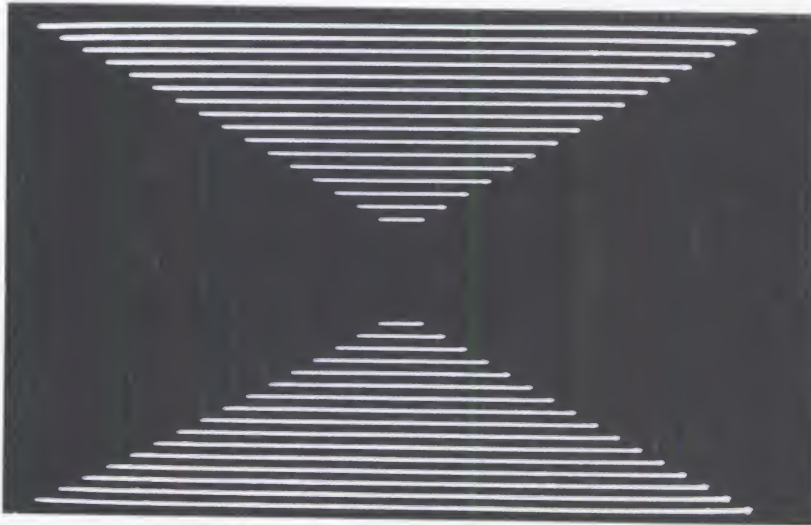
The program that follows draws two pyramids. One is right-side up and the other is upside down. Please type in the program:

```
10 GRAPHICS 14+16  
20 COLOR 1:SETCOLOR 0,8,12
```

```

30 A=0:B=191:C=159:D=0
40 FOR REPEAT=1 TO 16
50 PLOT A,D:REM 0,0
60 DRAWTO C,D:REM 159,0
70 PLOT C,B:REM 159,191
80 DRAWTO A,B:REM 0,191
90 A=A+5:B=B-5:C=C-5:D=D+5
100 NEXT REPEAT
110 FOR PAUSE=1 TO 100:NEXT PAUSE
120 GOTO 120
RUN

```



The first line is drawn from left to right at the top. This is controlled by lines 50 and 60 and the line is drawn from 0,0 to 159,0.

The second line is drawn from right to left at the bottom. It is controlled by lines 70 and 80 and it goes from 159,191 to 0,191. To see this clearly, type in:

```

85 FOR PAUSE=1 TO 200:NEXT PAUSE
RUN

```

Now you can see the lines being drawn one on top, then one on the bottom.

The keys to this drawing are lines 40, 100, and 90. Lines 40 and 100, as you know, tell the computer to repeat sixteen times whatever exists between the two lines. If you RUN the program again and count the number of lines drawn, you'll see there are sixteen lines for lines 50 and 60, and sixteen for lines 70 and 80.

When the computer reaches line 90, however, the values of A,B,C, and D change. Letters A and D have the number 5 added to them; B and C have the number 5 subtracted from them. So now:

13 GRAPHICS FOURTEEN

A	no longer equals 0.	A	equals 5 (A=0+5, OR A=5)
B	no longer equals 191.	B	equals 186 (B=191-5, OR B=186)
C	no longer equals 159.	C	equals 154 (C=159-5, OR C=154)
D	no longer equals 0.	D	equals 5 (D=0+5, OR D=5)

Line 100 sends the computer back to line 40 to repeat the process. The process will be repeated the number of times stated in line 40 (16). Since the values of A,B,C, and D have been changed, you can readily see that the lines will be drawn in different column and row coordinates. The second time through the loop, there is the equivalent of:

```
50 PLOT 5,5
60 DRAWTO 154,5
70 PLOT 154,186
80 DRAWTO 5,186
```

To repeat: due to lines 40 and 100, the computer will draw thirty-two lines. It will draw sixteen for lines 50 and 60, and sixteen for lines 70 and 80. Due to line 90, the lengths of the lines decrease.

Changing the values of variables is quite common in programming. To help you understand it thoroughly, I'll rewrite part of this program doing it the hard way—without the help of variables and line 90. If you understand the process, skip over this next part. Please *do not erase* the preceding program; it will be the basis for a number of other programs. Do *not* type in the following. It is merely a visual aid.

```
10 GRAPHICS 14+16
20 COLOR 1:SETCOLOR 0,8,12
30 REM THIS WOULD BE THE FIRST TIME THROUGH THE
  LOOP IN THE PRECEDING PROGRAM
40 PLOT 0,0:DRAWTO 159,0
50 PLOT 159,191:DRAWTO 0,191
60 REM THIS WOULD BE THE SECOND TIME THOUGH THE
  LOOP.
70 PLOT 5,5:DRAWTO 154,5
80 PLOT 154,186:DRAWTO 5,186
90 REM THIS WOULD BE THE THIRD TIME THROUGH THE
  LOOP.
100 PLOT 10,10:DRAWTO 149,10
110 PLOT 149,181:DRAWTO 10,181
120 REM FOURTH TIME.
130 PLOT 15,15:DRAWTO 144,15
140 PLOT 144,176:DRAWTO 15,176
150 GOTO 150
```

If you were to type in this program, you'd see the same first eight lines that are drawn in our first program. The difference (as you readily see) is that this program would have to be a lot longer than the first one to accomplish the same task. Line

90 saves a great deal of typing. Additionally, line 90 eliminates much adding and subtracting. If you have a repetitious job to do on your Atari, devise a program so the computer does it *for you*. Line 90 is a good example of the kind of tedious work the computer should take over.

Variation 1

How would you get the program to draw the pyramids, erase them, draw them again, etc.? Right! Change line 120 to:

```
120 GOTO 10
RUN
```

Now you see the program cycling.

Variation 2

The screen is not crowded at all. Let's add two more pyramids by changing just one number. How would you do this? Can you figure it out? Good! Change line 40.

```
40 FOR REPEAT=1 TO 32
RUN
```

There are four overlapping triangles and the program starts over and over again because of line 120.

Your LISTing should look like this:

```
10 GRAPHICS 14+16
20 COLOR 1:SETCOLOR 0,8,12
30 A=0:B=191:C=159:D=0
40 FOR REPEAT=1 TO 32
50 PLOT A,D
60 DRAWTO C,D
70 PLOT C,B
80 DRAWTO A,B
85 FOR PAUSE=1 TO 200:NEXT PAUSE
90 A=A+5:B=B-5:C=C-5:D=D+5
100 NEXT REPEAT
110 FOR PAUSE=1 TO 100:NEXT PAUSE
120 GOTO 10
```

(You may delete the pause loops, if you wish.)

Variation 3

Here's another change you can make:

```
60 PLOT C,D
80 PLOT A,B
RUN
```

13 GRAPHICS FOURTEEN

Instead of lines being drawn, you see individual dots of color. Change lines 60 and 80 again, please:

```
60 DRAWTO C,D
80 DRAWTO A,B
```

Variation 4

Let's have the background/border step through their colors. Can you do it without any help? Give it a try.

Here's one way. First, delete line 120, and add the following:

```
200 FOR CLR=1 TO 15 STEP 2:SETCOLOR 4,CLR,2:REM SETCOLOR
    4 CONTROLS THE BORDER/BACKGROUND COLOR
210 FOR WAIT=1 TO 150:NEXT WAIT:NEXT CLR
300 GOTO 10
RUN
```

The border and background are cycling through eight colors: 1, 3, 5, 7, 9, 11, 13, and 15. I put in the STEP 2 because the changes are more noticeable and dramatic this way.

Variation 5

Change line 300, but before RUNNING it, guess what will happen.

```
300 GOTO 200
RUN
```

Did you figure out what would happen? Good! The drawing stays on the screen while the background/border colors cycle forever...or until our next variation.

Variation 6

Now let's cycle the COLOR *and* the BRIGHTNESS of the background and border. Again, make the changes yourself before looking at my answer.

```
200 FOR CLR=1 TO 15 STEP 2 (OR NOT):FOR BR=0 TO 12 STEP
    3(OR NOT): SETCOLOR 4,CLR,BR
210 FOR WAIT=1 TO 150:NEXT WAIT:NEXT BR: NEXT CLR
RUN
```

(Of course the STEP increases and WAIT loop are optional.)

If you use the STEPs I did, you'll have eight colors with five brightnesses each, for a total of forty different color changes.

There are any number of additional changes you can make in this program. I suggest three:

- 1) You can control the color of the *drawing* cycle.
- 2) You can change the graphics mode and adjust the variables and number of repeats. The keys are still lines 40, 90, and 100. Here's one way to do this:

```

10 GRAPHICS 7+16
30 A=0:B=91:C=159:D=0
40 FOR REPEAT=1 TO 16
RUN

```

(Make any other graphics mode changes.)

- 3) Save the program. After you learn to do color in GR.15, change the program to GR.15 so you can make the two pyramids different colors.

Before you leave this program, I want to emphasize that the changing variable technique is probably the **single best way** to get striking drawings with a minimum of typing. Hard work, yes, but well worth it.

Program For Non-GTIA and Non-XL Owners

If you read the early part of this chapter, you know you can use the program for the "variable number of variables" technique in GR.7. (I hope you typed it in with the appropriate changes.)

The following program uses the "X+ or -, Y+ or -" technique. Type this in, please:

```

10 GRAPHICS 3+16:COLOR 1:X=5:Y=5
20 PLOT X,Y:REM 5,5
30 DRAWTO X,Y+13:REM 5,18
40 COLOR 2
50 PLOT X+5,Y:REM 10,5
60 DRAWTO X+5,Y+11:REM 10,16
70 COLOR 3
80 PLOT X+10,Y:REM 15,5
90 DRAWTO X+10,Y+9:REM 15,14
100 COLOR 1
110 PLOT X,Y+13
120 DRAWTO X+30,Y+13:REM 35,18
130 COLOR 2
140 PLOT X+5,Y+11:REM 10,16
150 DRAWTO X+30,Y+11:REM 35,16
160 COLOR 3
170 PLOT X+10,Y+9:REM 15,14
180 DRAWTO X+30,Y+9:REM 35,14
190 GOTO 190
RUN

```

With this method, X and Y stand for a number of things when this approach is used for drawing. The two can control the COLOR, BRIGHTNESS, and register numbers, as well as the column and row coordinates.

The first of these is the fact that the
the second is the fact that the
the third is the fact that the

the fourth is the fact that the
the fifth is the fact that the
the sixth is the fact that the

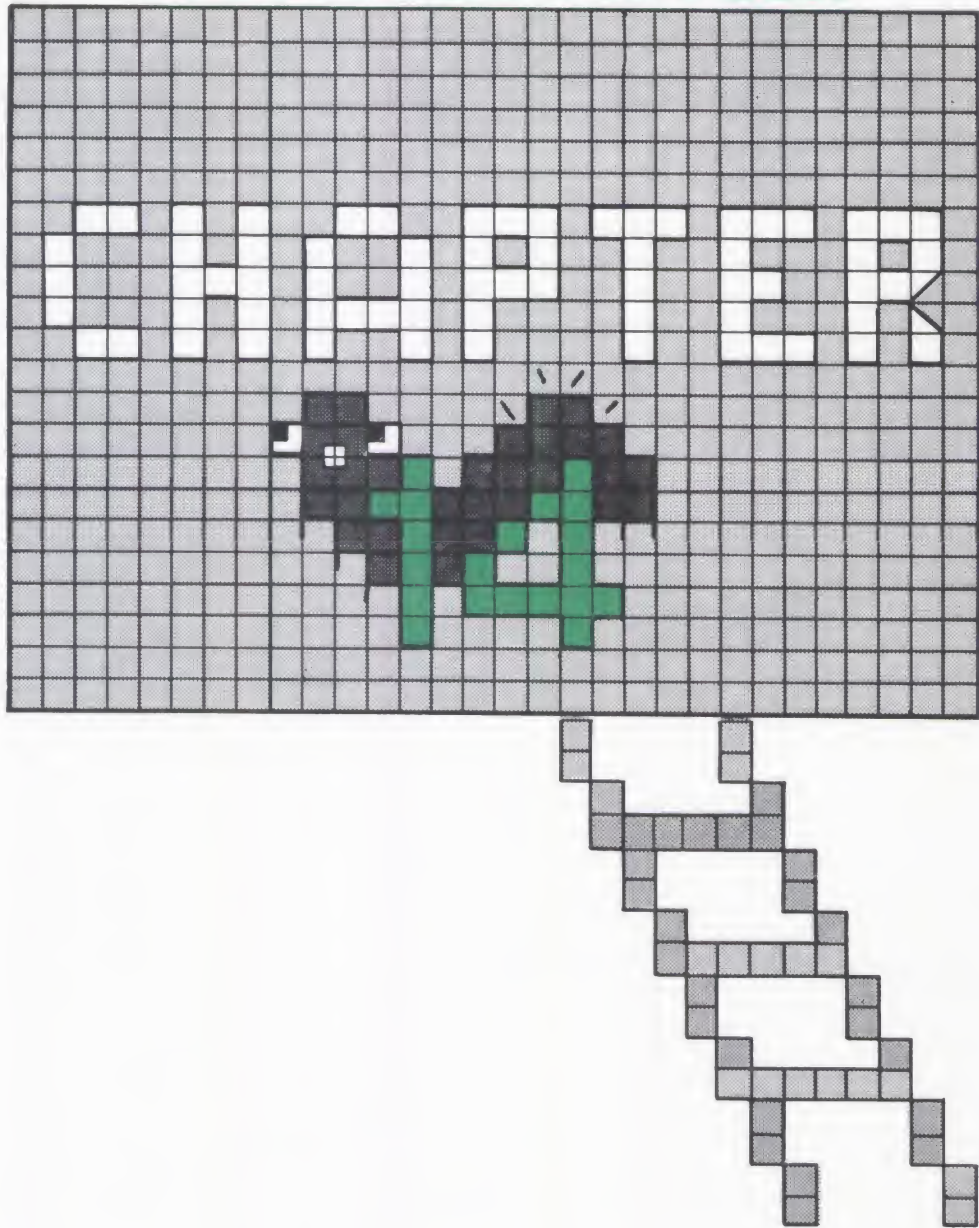
the seventh is the fact that the
the eighth is the fact that the
the ninth is the fact that the

the tenth is the fact that the
the eleventh is the fact that the
the twelfth is the fact that the

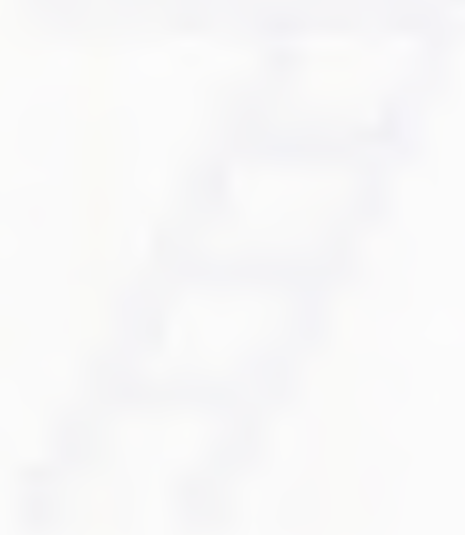
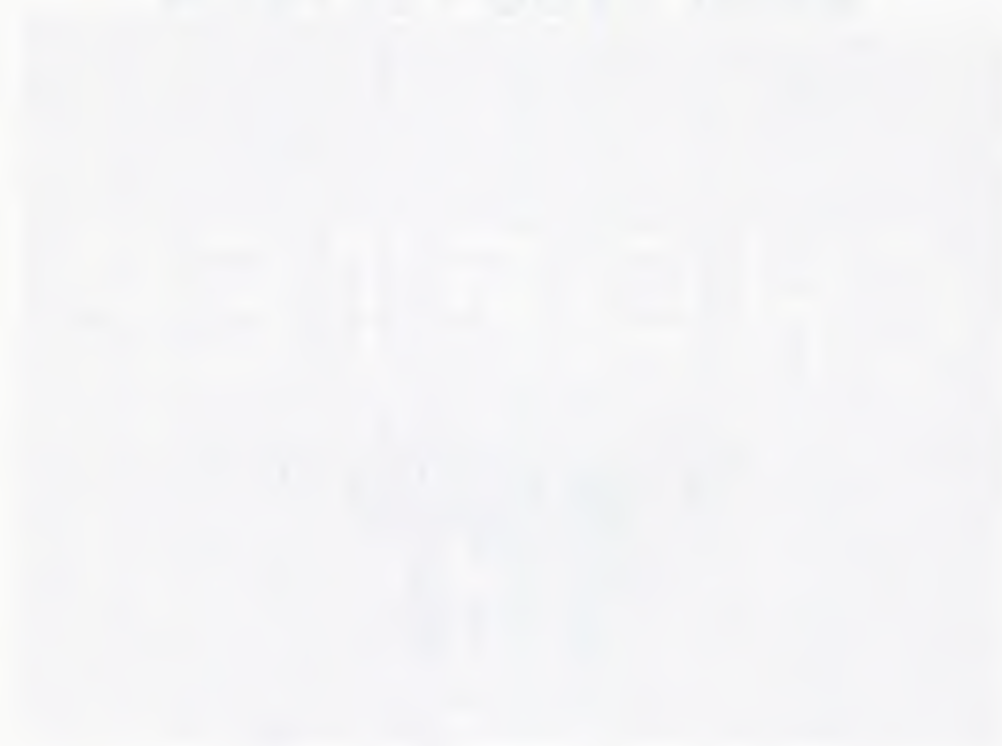
the thirteenth is the fact that the
the fourteenth is the fact that the
the fifteenth is the fact that the

the sixteenth is the fact that the
the seventeenth is the fact that the
the eighteenth is the fact that the

GRAPHICS FIFTEEN



RESEARCH REPORT



Graphics Fifteen

GR.15	—	160 X 160
GR.15+16	—	160 X 192

This is a four-color mode, including the border/background color.

Color Usage

COLOR 1 is orange. Change it by using SETCOLOR 0,C,B.

COLOR 2 is green. Change it by using SETCOLOR 1,C,B.

COLOR 3 is blue. Change it by using SETCOLOR 2,C,B.

Use SETCOLOR 4,C,B for the border/background.

Use SETCOLOR 2,C,B for the text window.

Please type in this easy program demonstrating the use of drawing colors in GR.15:

```

10 GRAPHICS 15+16
20 COLOR 1
30 PLOT 40,40
40 DRAWTO 120,40
50 COLOR 2
60 PLOT 40,80
70 DRAWTO 120,80
80 COLOR 3
90 PLOT 40,120
100 DRAWTO 120,120
110 GOTO 110
RUN

```

14 GRAPHICS FIFTEEN

This draws three lines across your screen in three colors. Make the following changes:

```
10 GRAPHICS 15+16:SETCOLOR 4,3,6
20 COLOR 1:SETCOLOR 0,5,14
50 COLOR 2:SETCOLOR 1,8,12
80 COLOR 3:SETCOLOR 2,1,10
RUN
```

You now have an idea of how to use the border/background color and the three foreground colors. You also see how to change the foreground colors. Ready for some tougher stuff?

What follows is a program that combines changing variables, a fast FOR...NEXT loop, and the XIO fill. It will draw four rectangles with connecting lines.

Note: Non-GTIA and non-XL owners should change line 10 to:

```
10 GRAPHICS 8+16
```

Type this in, please:

```
10 GRAPHICS 15+16
20 COLOR 1
30 REM ROW OF RECTANGLES
40 A=38:B=57:C=23:C=18
50 POKE 765,1
60 FOR RECTANGLE=1 TO 4
70 PLOT A,B:DRAWTO A,D:DRAWTO C,D:POSITION C,B
80 XIO 18,#6,0,0,"S:"
90 A=A+32:C=C+32
100 NEXT RECTANGLE
110 REM SWIFT FOR-NEXT LOOP
120 FOR A=35 TO 40
130 PLOT 0,A
140 DRAWTO 159,A
150 NEXT A
160 GOTO 160
RUN
```

I won't repeat how this is done, but will sum up the important parts. Lines 60 and 100 are the control; whatever falls between them is done four times. Lines 70 through 90 are responsible for drawing the four rectangles. The first rectangle is drawn with the coordinates of line 70; the next three are drawn at the same row coordinates, but at different column coordinates (A and C) due to the changing variables in line 90.

The next program has a total of twenty-two program lines instead of the sixteen, but it draws twelve rectangles and the connecting lines. It's helpful to

study it carefully and to RUN it. The fast FOR...NEXT loop is the most difficult to use with changing variables because you need *two* loops. One controls the actual FOR...NEXT loops for drawing, the other controls the number of repetitions. So, you have a nested FOR...NEXT loop with changing variables for the fast FOR...NEXT drawing loops.

(Note: Non-GTIA and non-XL owners should use GR.8+16 and COLOR 1 for all color statements.)

```

10 GR.15+16
20 REM DRAW TWELVE RECTANGLES
30 COLOR 1:POKE 765,1
40 A=32:B=57:C=23:D=18
50 FOR RECTANGLE=1 TO 12
60 IF RECTANGLE=5 THEN COLOR 2:POKE 765,
   2:A=32:B=B+58:C=23:D=D+58
70 IF RECTANGLE=9 THEN COLOR 3:POKE 765,
   3:A=32:B=B+58:C=23:D=D+58
80 PL.A,B:DR.A,D:DR.C,D:POS.C,B
90 XIO 18,#6,O,O,"S:"
100 A=A+32:C=C+32
110 NEXT RECTANGLE
120 REM DRAW THREE FAST FOR...NEXT LOOPS
130 E=35:F=40
140 FOR REPETITION=1 TO 3
150 IF REPETITION=2 THEN COLOR 2:E=E+58:F=F+58
160 IF REPETITION=3 THEN COLOR 3:E=E+58:F=F+58
170 FOR L=E TO F
180 PL.O,L
190 DR.159,L
200 NEXT L
210 NEXT REPETITION
220 GOTO 220
RUN

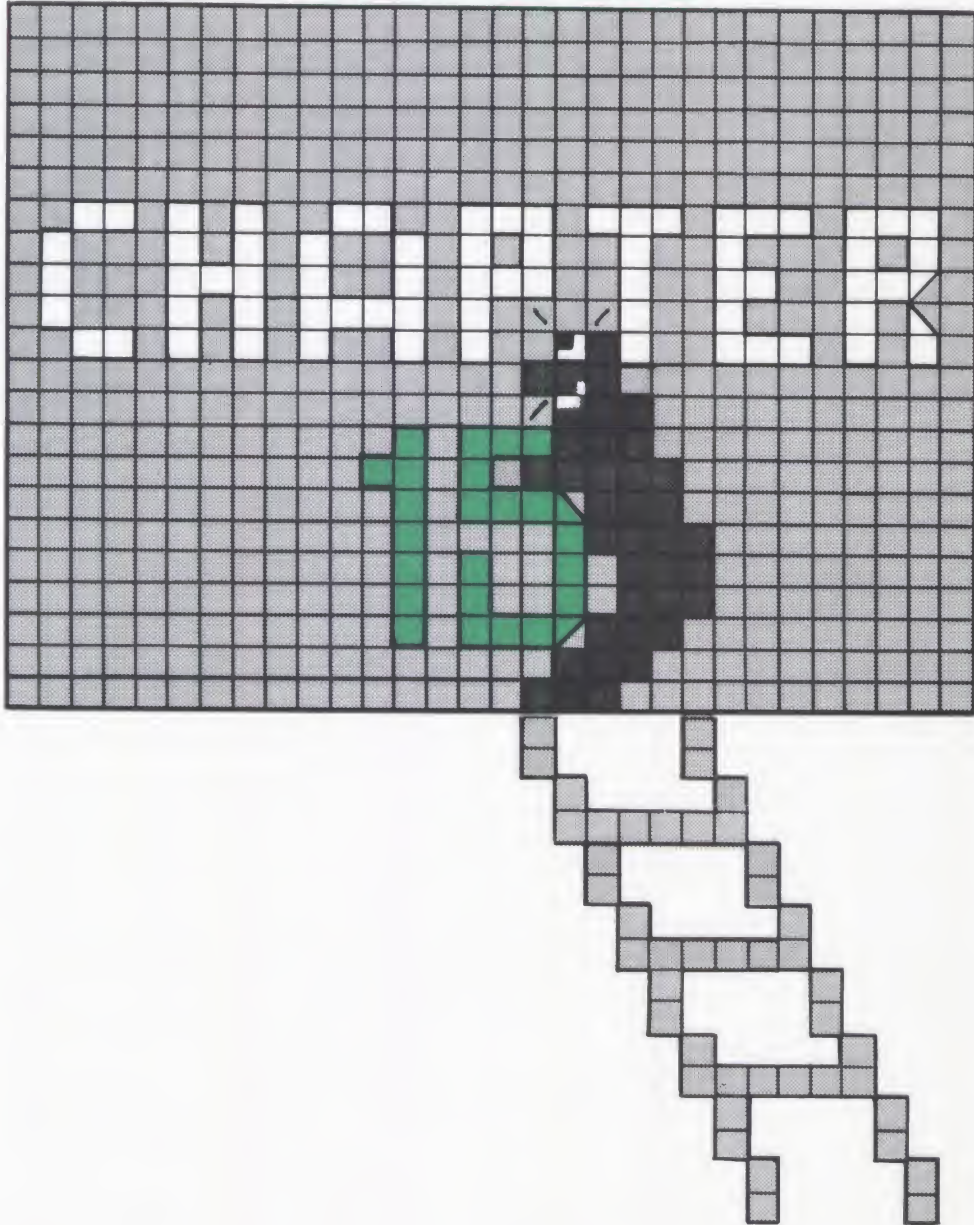
```

Since the non-GTIA and the non-XL owners can see this program in action, there is no separate section for them in this chapter.

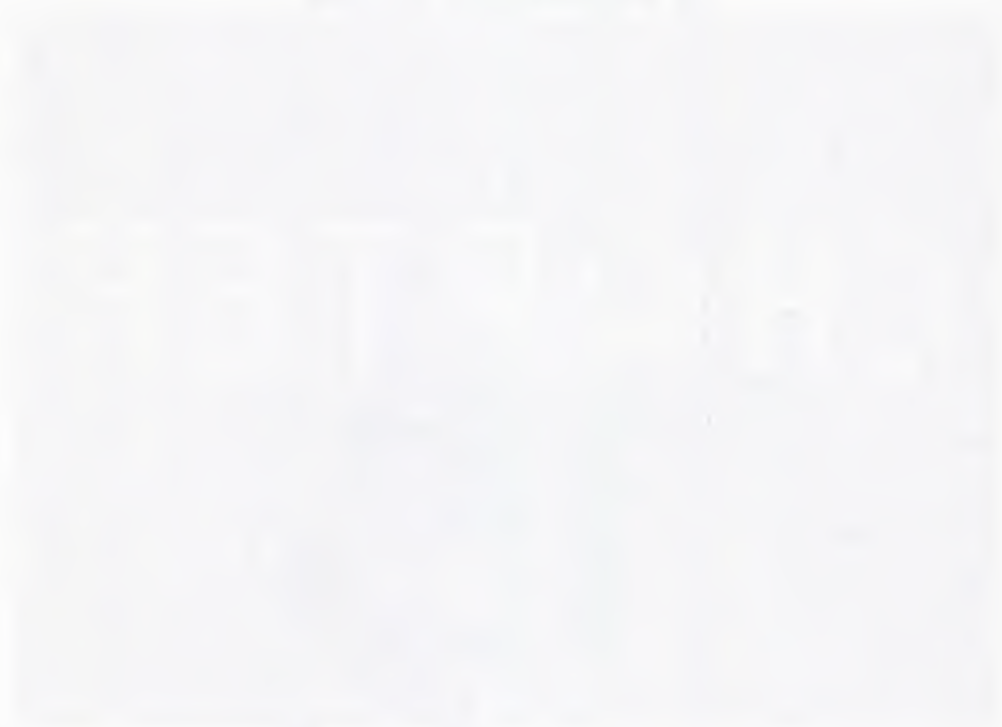
After studying the program, please choose another graphics mode and draw five squares in the same row numbers but at different column numbers. Draw lines through them with the fast FOR...NEXT loop and use the changing variable technique.

To add another change, use a STEP statement for the fast FOR...NEXT loop to draw lines every 2, 3, or 4 rows. You might also try using SETCOLOR statements and do some color cycling. Another thing to try is outlining the rectangles and connecting lines in a different color. Or, you can use triangles instead of squares. Keep experimenting; you learn more as you work more.

VARIABLES



21 FEB 1964



Variables

Some of the following programs use techniques we haven't discussed yet. If I were to explain these new concepts as I present the programs, it would be confusing, therefore let's examine the new techniques, first. I want you to see the larger picture—so please tolerate repetitious information about variables. My discussion will include numeric variables, string variables, DIMensioning, and the INPUT statement.

Numeric Variables

Type this in, please:

```
10 X=10
20 PRINT X
RUN
```

Your computer did not print the letter "X." Instead, it printed the number 10 because you told it that X equals 10.

Please type in the following:

```
10 FOR X=1 TO 100
20 PRINT X
30 NEXT X
RUN
```

Your computer did not print the letter "X;" rather, it printed the numbers 1 to 100 because you told it that X equals 1 to 100. In the above cases, X is called a variable with the values of 10 and 1 to 100, respectively. The variable X stores

15 VARIABLES

numbers, so it is called a numeric variable. You've used many numeric variables throughout this workbook: e.g., PAUSE, WAIT, WHOA, X, and Y. When you did use one, it was like creating a "box" inside of which is stored a number or numbers (the value).

Type in the following, please:

```
10 X=HORACE THINKLEBOOM
```

Your computer printed an "ERROR" sign as soon as you pressed the RETURN key. X, in this case, is a numeric variable and stores numbers only. You need another kind of variable to store things other than numbers.

String Variable

String variables will store numbers, letters, symbols, or a combination thereof. Please type in the following lines. ("X\$" is pronounced "ex string".)

```
10 DIM X$(20)
20 X$="HORACE THINKLEBOOM"
30 PRINT X$
RUN
```

If all went well, your computer printed the name.

Type this in, please:

```
10 DIM DATE$(30)
20 DATE$="AUGUST 9, 1938"
30 PRINT DATE$
RUN
```

Your computer printed the date string.

Essentially, you create a box in your computer's memory in which to store numbers (numeric variables). Or, you create a box to store numbers, letters, symbols, or a combination of the three. In this case, the box is called a string variable.

One requirement for a string variable is that the variable name must end with a dollar sign (\$): X\$, DATE\$, NAME\$, JV3M\$, etc.

DIMensioning

Another requirement for a string variable is that you must tell the computer how much space you want in the box. Do this with a DIMension (DIM) statement. You set the DIMensions in the previous programs with a DIM X\$(20) and DIM DATE\$(30). The boxes hold twenty characters in the first example and thirty in the second. This must be done before you can use a string variable. If you don't set aside enough space in the box, the computer will ignore the excess characters. For example, if you set aside twenty spaces but put twenty-five characters in the string, your Atari will ignore the last five characters.

The third requirement is the use of quotation marks around the value of the string variable. That's why you typed "HORACE THINKLEBOOM" and "AUGUST 9, 1938". You *must* use the quotes.

REVIEW

- 1) You must end the string variable with a dollar sign: e.g., ADD\$.
- 2) You must DIMension the length of the string variable: e.g., DIM ADD\$(20).
- 3) You must put the value inside quotation marks: e.g., "1345 L. STREET".

So far, we've talked about numeric and string variables where *you* assign the value—whether it's a number, a set of numbers, a name, an address, and so forth. But what do you do if you want the *user* of your program to enter the value of the variable?

INPUT Statement

Type this in, please:

```
10 PRINT "PLEASE TYPE IN YOUR AGE AND PRESS 'RETURN'."
20 INPUT A
30 PRINT : PRINT: PRINT: PRINT:REM CAUSES FOUR BLANK
    LINES ON COMPUTER SCREEN
40 PRINT A
RUN
```

Line 10 directed the computer to print the question. Then line 20 printed a question mark followed by the cursor, and waited for you to put in, or **INPUT**, your answer. This is one way the user assigns the value to a numeric variable. Notice that you do not have to DIMension a numeric variable.

Type in the following, please:

```
10 DIM ADD$(25)
20 PRINT "WHAT IS YOUR STREET ADDRESS"
30 INPUT ADD$
40 PRINT:PRINT:PRINT
50 PRINT ADD$
RUN
```

In this example, the computer asks for your address. Type it in, press RETURN, and your Atari stores the information in the string variable called ADD\$(ADD string). Later in the program (line 50), you tell the computer to print the contents of the box called ADD\$; it can because you set aside enough space in the box when you DIMensioned it.

The following program shows you how to set up your program for user INPUT into string variables, then PRINT the information the user provides.

15 VARIABLES

Type in this program, please:

```
10 DIM NAME$(35),ADD$(35),BIRTH$(20):REM NOTE THAT YOU
   USE A COMMA TO SEPARATE MORE THAN ONE DIM STATEMENT
20 PRINT "I'D LIKE TO OBTAIN SOME INFORMATION FROM YOU
   FOR MY FILES."
30 PRINT :PRINT
40 PRINT "PLEASE TYPE IN EACH ANSWER, THEN PRESS THE
   RETURN KEY."
50 PRINT :PRINT
60 PRINT "WHAT IS YOUR NAME":REM YOU DO NOT NEED A
   QUESTION MARK.
70 INPUT NAME$
80 PRINT :PRINT
90 PRINT "WHAT IS YOUR STREET ADDRESS"
100 INPUT ADD$
110 PRINT :PRINT
120 PRINT "WHAT IS YOUR DATE OF BIRTH"
130 INPUT BIRTH$
140 PRINT :PRINT
150 PRINT NAME$
160 PRINT
170 PRINT ADD$
180 PRINT
190 PRINT BIRTH$
200 GOTO 200
RUN
```

The computer asks questions, stores the answers in string variables, then releases the information according to lines 150, 170, and 190.

A word of caution at this point. When you DIMension a string variable, you're actually setting aside computer memory. Be careful *not* to be overly generous. For instance, don't DIM ADD\$(1000) because you'll be setting aside far too much memory. A good rule of thumb is to set aside a "1" for each letter, number, symbol, or space that you estimate will comprise the user's answer.

You've learned how to set up your program when you or a user put in the values of numeric or string variables and you know how to display that information. This brief introduction should help you understand the following programs. Study this chapter well and increase your efficiency. Two further suggestions before we move on: use the preceding program in GR.1 or 2 and add color.

MULTIPLICATION SLAVE





Multiplication Slave

Throughout this book, you've mainly learned to draw in color. In this section, you'll learn to design a program that's utilitarian rather than artistic. Someone will be *using* this program rather than looking at a picture; it's designed to drill the user on multiplication tables.

The program uses many of the techniques you've already learned and introduces some new ones. You'll find REMarks to help you understand what's going on. At the end, we'll discuss the program line-by-line.

Observe the number of lines that have multiple statements separated by a colon (:). In some cases this is done to conserve space. In others, the program doesn't work correctly unless the statements are in the same line.

Type in this program, please:

```

10 GRAPHICS 0
20 DIM ANSWER$(3),NAME$(20)
30 P=0:R=0:W=0
40 ? :? :? :? " (seven spaces) I FEEL YUCKY BECAUSE"
50 ? :? :? :? " (six spaces) I DON'T KNOW YOUR NAME."
60 ? :? :? :? " (six spaces) I WILL FEEL MUCH BETTER"
70 ? :? :? :? " (three spaces) IF YOU'LL INTRODUCE YOURSELF."
80 ? :? :? :? " (four spaces) PLEASE TYPE IN YOUR FIRST NAME."
90 INPUT NAME$
100 POKE 752,1
110 ? " (Press the ESC key and release it. Press the
    CTRL/CLEAR keys, then release them.) ":? :? "THANKS,
    (one space) ";NAME$;"."
```

16 MULTIPLICATION SLAVE

```

120 ? :? :? :? :? :? :? :? " (ten spaces) I FEEL MUCH BETTER"
130 ? :? :? :? :? :? :? :? " (six spaces) NOW THAT I KNOW
    YOUR NAME."
140 FOR X=1 TO 1500:NEXT X
150 ? "(press ESC, CTRL/CLEAR) "
160 ? :? :? :? :? :? :? :? " (five spaces) IF YOU WOULD LIKE
    TO SEE"
170 ? "(five spaces) THE INSTRUCTIONS, TYPE Y."
180 ? :? :? :? :? " (five spaces) IF YOU WOULD NOT LIKE TO SEE"
190 ? "(five spaces) THE INSTRUCTIONS, TYPE N."
200 INPUT ANSWER$
210 IF ANSWER$="N" THEN 310
220 ? "(ESC, CTRL/CLEAR) ":? :? " (eight spaces) WHEN
    YOU SEE A PROBLEM"
230 ? :? :? " (twelve spaces) ON THE SCREEN,"
240 ? :? :? " (nine spaces) TYPE IN THE CORRECT"
250 ? :? :? " (ten spaces) NUMBER OR NUMBERS."
260 ? :? :? " (six spaces) THEN PRESS THE RETURN KEY."
270 ? :? :? " (twelve spaces) WHEN YOU HAVE"
280 ? :? :? " (five spaces) DONE ENOUGH PROBLEMS, (one space)
    ";NAME$;"",
290 ? :? :? :? " (five spaces) TYPE IN -1 AND PRESS RETURN."
300 FOR X=1 TO 3500:NEXT X
310 ? "(ESC, CTRL/CLEAR) ":? :? :? :? :? :? :? :? :? :?
    :? :? :? " (four spaces) LET'S BEGIN THE PROGRAM, (one
    space) ";NAME$;"!"
320 FOR X=1 TO 800:NEXT X
330 GRAPHICS 2+16:POKE 752,1
340 POSITION 3,4:? #6;" (Make the odd-numbered letters upper
case, and the even-numbered letters lower case. One way to do this is
to type M L I L C T O then use the CTRL/left arrow key to return to
the space between the M and L, then press the CAPS LOWR key and
type in the u. Use your CTRL/right arrow keys to move the cursor to
the space between the L and the I, type in the t, and continue until
you have finished typing the word.) MuLtIpLiCaTiOn (Press the
SHIFT/CAPS LOWR keys at the same time.) "
350 POSITION 7,8:? #6;" (Press the DOM key, then type S A E with
a space between each letter. Use the CTRL/left arrow keys to return
to the space between the S and the A. Press the CAPS LOWR key,
type in the l. Use the CTRL/right arrow keys to move the cursor to
the space between the A and the E and type in the v.) SlAvE (Press
the DOM and SHIFT/CAPS LOWR keys. Use CTRL/right arrow
to move one space.) "
360 FOR X=1 TO 1500:NEXT X

```


Program Lines for XL Owners

```
370 GR.14+16:REM ROUTINE FOR READY!
380 REM SET BORDER/BACKGROUND COLOR.
390 SE.4,12,14
400 REM SET FOREGROUND COLOR
410 COLOR 1:SE.0,6,2
```

Program Lines for Non-XL Owners

```
370 GRAPHICS 8+16:REM ROUTINE FOR READY!
380 REM SET THE BORDER/BACKGROUND COLOR.
390 SETCOLOR 4,10,14:SETCOLOR 2,10,14
400 REM SET FOREGROUND COLOR.
410 COLOR 1:SETCOLOR 1,6,2
```

Program Lines for Both XL and Non-XL Owners

```
420 REM INSTRUCTIONS FOR DRAWING READY!
430 REM R
440 PLOT 20,170:DRAWTO 20,25:DRAWTO 39,25:DRAWTO
    39,95:DRAWTO 20,95:PLOT 22,96:DRAWTO 36,170
450 REM E
460 PLOT 64,170:DRAWTO 45,170:DRAWTO 45,25:DRAWTO
    64,25:PLOT 45,95:DRAWTO 64,95
470 REM A
480 PLOT 70,170:DRAWTO 70,25:DRAWTO 89,25:DRAWTO
    89,170:PLOT 70,95:DRAWTO 89,95
490 REM D
500 PLOT 95,25:DRAWTO 108,25:DRAWTO 114,60:DRAWTO
    114,135:DRAWTO 107,170:DRAWTO 95,170:DRAWTO 95,25
510 REM Y
520 PLOT 120,25:DRAWTO 120,50:DRAWTO 129,95:DRAWTO
    129,170:PLOT 139,25:DRAWTO 139,50:DRAWTO 129,95
530 REM !
540 PLOT 150,25:DRAWTO 150,160:PLOT 150,166:DRAWTO
    150,170
550 FOR X=1 TO 200:NEXT X
560 REM USE GRAPHICS TWO FOR DRILL
570 GRAPHICS 2:POKE 752,1:REM SET GRAPHICS MODE AND
    SUPPRESS CURSOR.
580 REM FORMULAE FOR CHOOSING RANDOM NUMBERS FOR THE
    MULTIPLICATION PROBLEMS.
590 A=INT(12*RND(1))+1
600 B=INT(12*RND(1))+1
610 REM PLACE MULTIPLICATION PROBLEMS AT A CERTAIN
    LOCATION ON THE SCREEN.
620 POSITION 7,3
```

16 MULTIPLICATION SLAVE

```

630 REM TELL COMPUTER HOW TO DISPLAY THE PROBLEMS.
640 ? #6;A;"*";B;"="
650 INPUT N:REM THE QUESTION MARK AFTER WHICH THE USER
    WILL TYPE IN HIS ANSWER.
660 IF N=-1 THEN 740
670 IF N<>A*B THEN POSITION 1,6:? #6;"SORRY. TRY
    AGAIN.":P=P+1:W=W+1:GOTO 650
680 E=INT(4*RND(1))+1:P=P+1:R=R+1
690 IF E=1 THEN POSITION 1,6:? #6;" (six spaces. DOM) G
    (DOM. CAPS LOWR) o (SHIFT/CAPS LOWR. DOM) O
    (CAPS LOWR) d (SHIFT/CAPS LOWR. DOM. TYPE
    EXCLAMATION POINT) ! (nine spaces) ":FOR X=1
    TO 200:NEXT X:GOTO 570
700 IF E=2 THEN POSITION 1,6:? #6;" (six spaces. CAPS LOWR)
    right (SHIFT/CAPS LOWR. seven spaces) ":FOR X=1
    TO 200:NEXT X:GOTO 570
710 IF E=3 THEN POSITION 1,6:? #6;" (three spaces) YOU (one
    space. DOM) GOT (DOM. one space) IT! (five spaces)
    ":FOR X=1 TO 200:NEXT X:GOTO 570
720 IF E=4 THEN POSITION 1,6:? #6;" (CAPS LOWR. two
    spaces) cor (DOM. SHIFT/CAPS LOWR) REC (DOM. CAPS
    LOWR) ta (SHIFT/CAPS LOWR) MUN (CAPS LOWR) do
    (SHIFT/CAPS LOWR)!! (one space) ":FOR X=1 TO
    200:NEXT X:GOTO 570
730 REM ELIMINATE TEXT WINDOW.
740 GRAPHICS 2+16:POSITION 1,2:? #6;" (DOM. CAPS LOWR)
    no. of problems= (DOM. SHIFT/CAPS LOWR) ";P
750 POSITION 1,4:? #6;" (CAPS LOWR) no. incorrect=
    (SHIFT/CAPS LOWR) ";W
760 IF W=0 THEN POSITION 1,8:? #6;"GREAT, (one space)
    ";NAME$;"!":POSITION 1,10:? #6;"NONE WRONG!"
770 REM NEXT TWO LINES FIGURE THE PERCENTAGE OF PROBLEMS
    CORRECT.
780 PERCENT=R/P
790 PERCENT=INT(100*PERCENT+0.5)
800 POSITION 1,6:? #6;" (DOM) PERCENTAGE= (DOM. one space)
    ";PERCENT;"%"
810 FOR X=1 TO 1500:NEXT X:GRAPHICS 2+16
820 FOR J=0 TO 14 STEP 2:SETCOLOR 4,J,8:FOR L=1 TO 50:NEXT
    L:NEXT J
830 GRAPHICS 2+16:POSITION 1,1:? #6;" (five spaces) I'M PROUD"
840 POSITION 1,3:? #6;" (one space) OF YOU, (one space)
    ";NAME$;"", "
850 POSITION 1,5:? #6;"FOR WORKING ON YOUR"
860 POSITION 3,7:? #6;" (DOM. CAPS LOWR) mul
    (SHIFT/CAPS LOWR) TI (DOM) PLI (CAPS LOWR) ca
    (SHIFT/CAPS LOWR) TION."

```



```

870 POSITION 1,9:? #6;" (three spaces) YOU'LL BE GLAD"
880 POSITION 1,11:? #6;" (six spaces) YOU DID."
890 FOR X=1 TO 100:NEXT X:GOTO 830
RUN

```

You may adjust the pause loops right away; they are annoying if they're too slow.

Setting up a program forces you to make artistic choices. I used color only for the main part of the program to make it stand out after the drab, colorless introduction. To my mind, the user sees the color and it triggers the psychological response: "Okay, now the program *really* starts."

Line-by-Line Program Description

- Line 10 sets the graphics mode.
- Line 20 DIMensions the two responses for the user. Notice the arrangement. If you DIMension more than one variable, use the DIM statement only once, then separate the DIMensioned variables with a comma.
- Line 30 initializes **P**, **R**, and **W**. This means it gives the initial, or beginning, value to each variable. The letter **P** stands for the number of problems the user answers; the **R** stands for the number of problems he gets right; the **W** is for the number of wrong answers.
- Lines 40
through 80 "personalize" the computer.
- Line 90 obtains the user's name.
- Line 100 suppresses the cursor.
- Line 110 clears the screen. When you press the ESCape key, release it, then hold down the CTRL and CLEAR keys simultaneously, you will clear the screen at this point in your program. You must follow the sequence ?"(ESC, CTRL/CLEAR)" however, to delay the execution of the statement until the computer reaches this line in your program.

In this line, a colon (:) follows the instructions to clear the screen and the computer knows another statement will follow. There is a ? statement after the colon telling the computer to move down one line on the new (cleared) screen. The ? is followed by another colon, again telling the computer that another statement follows.

The next part of line 110 directs the computer to print something on the screen. The statement is a little more complicated than the usual PRINT statement because it includes the name of the user (which has been stored in the string variable name since line 90).

16 MULTIPLICATION SLAVE

The semicolons direct the computer to stay on the same line.

The computer prints the information between the quotation marks (it prints **THANKS**, and a period). Between these two items, the computer will dump the contents of the string variable NAME\$. It does so on the same line because of the semicolons. Thus you have the word **THANKS** followed by a comma, then by the contents of the variable NAME\$, followed by a period. (If you don't put in the space after the comma, the computer will place the string variable NAME\$ contents right after the comma.)

- Lines 120
and 130 are similar to lines 40 through 80. I decided to skip lines and center the message to make the display visually more pleasing.
- Line 140 is a pause loop.
- Line 150 clears the screen.
- Lines 160
through 200 are a convenient option. After the user learns a program, you don't want to force him to sit through the instructions each time. By providing this option, you let him skip to the meat of the program. He will type in either a *Y* or an *N* (Line 200).
- Line 210 instructs the computer to skip line 310 if the user types in an *N* (meaning he does *not* want to see the instructions). If the INPUT in line 200 is *Y*, line 210 is ignored and the instructions will be displayed.
- Lines 220
through 290 clear the screen and tell the user how to interact with the program.
- Line 300 is a pause loop.
- Line 310 clears instructions from the screen, centers the message, and prints the user's name.
- Line 320 is a pause loop.
- Line 330 directs the computer to use GRAPHICS 2+16 for the title and suppress the cursor.
- Lines 340
and 350 directs the Atari to use upper case, lower case, DOM key, and combinations to print the title in colors.
- Line 360 is a pause loop.
- Lines 370
through 410 set the graphics mode and color information.
- Line 420 is a REMark.

Lines 430 through 540 provide instructions for printing the "READY" on the screen.

Line 550 is a pause loop.

Line 560 is a REMark.

Line 570 sets the graphics mode and suppresses the cursor.

Line 580 is a REMark.

Line 590 tells the computer that A is a random number between 1 and 12.

Line 600 tells the computer that B is also a random number between 1 and 12. A and B will be the two parts of each multiplication problem.

Line 610 is a REMark.

Line 620 is the position of the problem on the screen.

Line 630 is a REMark.

Line 640 tells the computer how to display the problem.

Line 650 accepts the user's answer to the problem; it concludes with a REMark.

Line 660 examines the user's INPUT. In the instructions, the user was told to type in -1 in order to stop doing problems. If he indeed has typed it in, the computer skips to line 740.

Line 670 checks to see if the INPUT for the problem is incorrect. If the user's INPUT is *not* -1, line 660 will be ignored. If the user's INPUT (N) does not equal (< >) the answer to the problem (A times B), the Atari prints the message, "SORRY. TRY AGAIN."

After the message is printed, the computer adds 1 to both the values of **P** and **W**. Since we set the values of **P** and **W** at 0 (line 30), their values have now increased until **P** equals 1 (**P**=1) and **W** equals 1 (**W**=1). **P** stands for the number of problems; **W** stands for the number of wrong answers. The computer adds 1 to the values of **P** and **W** each time the user makes a mistake in his multiplication.

After the computer adds 1 to the values of **P** and **W**, it will GOTO 650, meaning it will print another question mark (?) and wait for the user's next answer to the same problem.

Line 680 is related to lines 650 through 670 (they are the keys to understanding line 680). If you don't comprehend the logic of the program at this point, go back to the beginning and re-trace it line-by-line.

In line 650, the user types in something and the computer *must* respond. It will either move to 740 (line 660), or it will print a message, add 1 to the values of **P** and **W**, and return to line 650 so the

16 MULTIPLICATION SLAVE

user can again try to answer the problem correctly (line 670).

The Atari makes decisions using the IF...THEN format. IF the INPUT isn't satisfactory for lines 660 and 670, it continues to 680, which should be viewed as the third alternative.

To put it another way, the computer has three choices after the user's INPUT. It can stop the flow of problems, or indicate a wrong answer, or continue to line 680. IF it continues, THEN the user did *not* type -1, and he did *not* make a mistake in his multiplication.

In line 680, the computer is told to pick a random number between 1 and 4 and assign that number to the variable **E**. It is also directed to add 1 to the values of **P** and **R**. Thus the computer will add 1 to the number of problems and 1 to the number of right answers. Since there are no further instructions, the Atari moves on.

Lines 690

through 720 offer four random responses when the user types in a correct answer. The Atari has chosen a random number between 1 and 4 (line 680), and has placed it in variable **E**. Let's assume the computer has chosen the number 4; **E**, therefore, is equal to 4 ($E=4$).

When the right answer is entered, the subsequent response can be "GOOD," "RIGHT," "YOU GOT IT," or the Fonz's favorite, "CORRECTAMUNDO." One of these four responses will be printed if **E** equals the correct number. IF $E=1$, THEN the computer will print "GOOD." IF $E=2$, THEN the computer will print "RIGHT," and so forth.

The random number chosen by the computer in line 680 determines which message is printed. After a message is printed, *regardless* of what the message is, the Atari pauses so the user has time to read the message. Then it will GOTO 570 which clears the screen and places another problem there.

If we assume, then, that the user doesn't type in -1 or an incorrect answer, the computer will continue randomly to print one of the four congratulatory messages and will place another problem onscreen until the bitter end—of the Atari or the user!

Line 730

is a REMARK.

Line 740

is linked to line 660. IF the user's INPUT is -1 in line 650, THEN the computer goes to line 740 because of line 660. It removes the text window at the bottom of the screen, POSITIONs three words, and has the computer print the number of problems (**P**) that the user has answered.

Line 750

directs the computer to POSITION some words and to print the number of incorrect answers (**W**).

- Line 760 prints a congratulatory message if $W=0$. Recall that **W** stands for the number of wrong responses; the computer prints this message *only* if the user has made no mistakes ($W=0$). It will also print the user's name.
- Line 770 is a REMark.
- Line 780 tells the computer that the number of right answers divided by the number of problems is the percentage.
- Line 790 causes the Atari to round off the percent figured in line 780, so we have only whole numbers, or integers, for the number printed in line 800.
- Line 800 asks the computer to print **PERCENTAGE=** and then put in the percent computed in line 780 and rounded off in line 790.
- Line 810 is a pause loop. The $GR.2+16$ clears the screen.
- Line 820 sets the **COLOR**, pauses, then cycles the **COLOR**.
- Line 830 contains the **GR**aphics statement which clears the screen. Also, the beginning of the message is printed.
- Lines 840 through 880 print the rest of the final message for the user.
- Line 890 is a pause loop; the computer then returns to line 830. Due to the graphics statement there, the screen is momentarily cleared. This gives the message a "blinking" effect.

Suggestions For Non-XL Owners

You may want to center the "READY" statement. Since it was designed for $GR.14+16$, it's set on a 160 by 192 grid. You may also change the mode from 8 to 7, to 3, or whatever.

Suggestions For XL and Non-XL Owners

Consider using happy and sad faces for two additional responses to the user's **INPUT** after a problem. You'll probably use a **GOSUB** and **RETURN**, or you might have to use **GOTO** statements. In any event, be careful. Before you attempt anything, study the four other responses and how they're set up.

Further Suggestions

Why not try to:

- 1) begin the program in color
- 2) change the graphics modes
- 3) put the entire program in **GRAPHICS 0**

16 MULTIPLICATION SLAVE

- 4) pare down the program to its bare bones so it starts with a problem. Include no instructions, title, colors, etc. Make it an unexciting, utilitarian program; then re-work it, adding your choices for color, mode, etc.
- 5) set up the program so it will drill the user on the seven through twelve tables instead of the one through twelve
- 6) change the responses completely for the messages to the user
- 7) change the color of the text window for the user's INPUT
- 8) put in a stunning color/graphics demonstration, or
- 9) change the program to a foreign language.

Explore the limits of the program. Ask yourself, "What if ...?" and search for new ways to modify the program. The more you study and practice changing it, the more you'll learn.

Afterword

I hope you've learned many new things from this book and that it hasn't been too difficult for you. I've enjoyed making up the programs and explaining them in what I hope is a helpful manner.

You now know the basics of programming and how to use color. I sincerely hope you continue to learn more about your Atari.

To further your education, check out the following:


- 1) Sound dresses up programs nicely; it's a worthwhile addition.
- 2) Redefining character sets is difficult, but the results are exceptional.
- 3) Player/missile (P/M) graphics are what most programmers want to use, but they require a good deal of knowledge and patience to master.

Much of the software on the market today enables you to use your Atari with a minimum knowledge of computers. For instance, there are programs that do most of the work to redefine character sets and use P/M graphics. Unfortunately, if you were to use software to do one of the aforementioned tasks, you have to learn much more to use it in a program. After all, you're the one—the *only* one—giving the commands.

Obviously, a great deal remains for you to learn about programming. You're off to a good start, though, so have fun and good luck!

Glossary**

Artifacting Illuminating two side-by-side pixels in GR.8 will give you one color. However, illuminating only the odd-numbered, or only the even-numbered pixels, results in two other (different) colors. When you add these to one color each for the background and the border, you wind up with five colors that may be placed on the screen in this graphics mode. Artifacting is the name for this unusual approach to placing color onscreen.

Atari Logo Key () Special key on the Atari computer keyboard that, when pressed, causes the character set to be displayed in inverse video on the monitor.

Bit Acronym for BInary digiT. A bit is either a 0 or a 1. Grouped bits are used in computer systems to code information, instructions, and data. A larger unit of bits is the byte (eight bits).

Border Refers to the area of your TV or monitor screen that surrounds the background. The border color is programmable using the BASIC SETCOLOR or POKE commands. You may experiment with various border colors in GR.0 by POKEing any integer from 0 to 255 into location 712.

BREAK Key To end or break the current function in any program, at any time, press the BREAK key.

Brightness See Luminance.

Bug An error in the processing logic of a program. See Debug.

Byte A measurement of storage capacity in a computer's memory. A byte is a line

or row of eight bits that is treated as a unit. In graphics modes, it takes eight bytes to store the instructions that form a character or a symbol on the screen.

Character Graphics A term that describes the use of control characters and/or redefined characters to create a graphics display. Many of the high-resolution graphics displays in game programs employ character graphics.

COLOR Atari BASIC command. The COLOR command does not change any of the color registers, but is used to define the data to be used by commands, such as PLOT and DRAWTO, that follow. When used in a graphics window of a screen, COLOR defines the color data that will appear on the screen. When used in a text display mode, COLOR defines the character that can be plotted or drawn onscreen.

Color Clock The color clock is a measure or unit of horizontal distance. In GR.8, each pixel is half a color clock wide, meaning that two pixels, illuminated side-by-side, give you one color.

Color Register The value stored in each color register allows you to place color on the screen. The Atari has nine color registers, stored within the computer's operating system. Four are used for player/missile graphics, an advanced technique. Five registers remain, but not all of them can be used in all of the graphics modes.

Column The line of invisible boxes stacked vertically in a grid. The X-coordinate of a grid.

Command A statement entered by the user that causes a computer to carry out a specific action. Commands are usually acted upon immediately by the computer.

Control (CTRL) Key The CTRL key is used in combination with other keys on the Atari keyboard, giving them added functions.

Control Characters Characters with specific system-dependent meanings. Many keys on your computer keyboard have graphics characters you can use. Use them by pressing the CTRL key at the same time you press a letter key which has a graphics character beneath it.

Cursor The cursor is the square symbol that appears on the screen to let you know where an action (such as typing in a character, deleting, inserting, etc.) will take place.

Cursor Control Keys Keys used with the CTRL key to position the cursor (e.g., the up arrow key). Most cursor control keys are defined by the currently executing program, and not by the Atari's hardware.

Debug Test a program. A programmer must make sure that a program will correctly process all of the types of data for which it is intended. Samples of the data are prepared (test data), and the program is executed using it (a test run). The program's outputs (reports, files, screen displays, etc.) are then verified to be as

specified. An error in the processing logic of a program is called a "bug," hence the terms "debug" and "bug-free."

Default A value for a parameter or variable which is supplied by a system if the user does not specify an explicit value.

Default Mode In Atari color graphics, GR.0 is considered the default mode since the computer returns to it after a program is finished (unless the user has entered an instruction, such as GOTO).

DIMension Atari BASIC statement. Before they can be used, all string variables must be DIMensioned. DIM sets aside space for a "box" in computer memory that will hold the string variable.

DOM Key Special key on the Atari computer keyboard that, when pressed, causes the character set to be displayed in inverse video on the monitor.

DRAWTO Atari BASIC command that causes a line to be drawn from the last point of the previous DRAWTO or PLOT command to the point specified after DRAWTO.

FOR Atari BASIC command. This command, together with TO, STEP, and NEXT, sets up a loop in a program and tells the computer how many times to perform the loop before proceeding to the next command.

GOSUB Atari BASIC command. A consecutive series of commands to compute a certain value, print a particular message on the screen, etc., is called a subroutine. When a subroutine is needed several times in a program, you can save time and memory by not rewriting it. The GOSUB statement allows you to write the subroutine once and call it up for processing at any necessary point in the program. A GOSUB may be placed anywhere in the program. Using this procedure makes programs easier to read and debug.

GOTO Atari BASIC statement. GOTO, followed by a number, branches the program to the number. The program then continues by consecutive line numbers unless branched again.

GRAPHICS Atari BASIC command. When followed by a number, it is used to select one of the Atari graphics modes. The text window can be eliminated if +16 is added to the GR. number for modes 1 through 8. GR. is the abbreviation for GRAPHICS.

Graphics Mode A form or variety of screen display. An invisible grid made up of a number of boxes arranged in columns and rows on the computer monitor or TV screen. There are up to sixteen different varieties of screens (grids) available to the user, depending upon your model Atari. Some modes are text modes; others are used for drawing (though some of these provide you with four lines of text mode at the bottom of the screen, known as a text window).

Grid The invisible "checkerboard" on the TV or monitor screen, that's used for text or graphics (depending on the graphics mode chosen by the user). Grids are

comprised of lines of boxes arranged in columns (the X-coordinates) and in rows (the Y-coordinates). Grids vary as to the total number of boxes in their respective columns and rows. In the different text modes, different grids will display characters of different sizes according to the variation in the number of boxes they contain.

High-Resolution Graphics Resolution is the total number of pixels a computer uses to display a picture onscreen. The higher the number of pixels, the higher the resolution; the higher the resolution, the more detail a picture can have. Pictures made on a screen using high-resolution are termed high-resolution graphics.

IF Atari BASIC statement. IF sets the conditions for the IF...THEN conditional branch statement; THEN introduces the consequential commands that are executed only if those conditions are true. Whether the branch is executed or not, the program goes to the next numbered line.

Initializing a Variable In programming, initializing occurs when you give a variable within a line its initial (i.e., beginning) value.

INPUT Atari BASIC command. When INPUT followed by a variable name is executed by the computer, a "?" is printed onscreen and the program stops. The user must then type in the number or string previously requested and press RETURN. After this is done, the program stores the response in the selected variable name and continues with the next command after the INPUT. When the INPUT is for a string variable, the variable must have been DIMensioned previously.

Inverse Video Using one of three special keys (Atari logo, DOM, or semaphore) on the computer's keyboard, the video monitor can be reversed so that light characters and/or symbols appear on a dark background.

Kilobyte A byte is a measurement of storage capacity in computer memory. A thousand bytes is a K byte, abbreviated KB, or K. Actually, one kilobyte equals 1,024 bytes, an even power of 2.

Limiter The controller, or limiter, in nested FOR...NEXT loops is the outside loop.

LIST Atari BASIC command. With no line number specified, LIST outputs all lines of the program in memory (in numerical order) to the devices specified. If a line number is given, only that line will be output. To get a range of lines, specify the first and last line numbers, separated by a comma. A program, or any part thereof, may be listed to a screen, printer, disk, or cassette.

LOAD Atari BASIC command that causes the tokenized version of a program on disk to be put into the computer's memory.

Loop, Delay A delay loop is used to freeze the screen briefly while the operator using your program reads a message.

Loop, Nested Loops may be nested within one another. In such cases each loop will be completed before the next one is started, beginning with the innermost and progressing to the outermost loop.

Loop, Wait Typing the word WAIT in a program line slows down the execution of the program considerably.

Luminance Atari's term for brightness. In the BASIC SETCOLOR command, the third operand is the luminance. Eight luminances (numbered from 0 to 14, even numbers only) are available for use in color graphics. The number 14 provides maximum brightness, while 0 provides deep, dark color.

Memory Any device that can store information for retrieval when needed. Microcomputers rely primarily on random access memory (RAM), read-only memory (ROM), floppy disks, hard disks, and cassettes. ROM and RAM together make up the internal, or main, memory of the computer. Data in internal memory is immediately available to programs for processing. External memory devices, such as cassettes and diskettes, require mechanical motion to retrieve data. By itself, the term memory usually refers to RAM.

Memory Address A number or variable designating a location in memory.

NEW Atari BASIC command. When starting programs, NEW is used to erase all the lines of old programs in memory. It clears all memory and closes all files without turning off the computer. Press RETURN after NEW is typed in; NEW is generally used when you are finished with a piece of work and are ready to start a new program.

NEXT Atari BASIC command. NEXT, the last statement of the FOR...NEXT loop, determines whether the loop continues or terminates, depending on the range of index numbers set up for the loop.

Numeric Variable A numeric quantity whose value may, but does not necessarily have to, change. Atari BASIC has a limit of 128 variable names per program. See Variable.

Operating System The software that manages the computer's hardware and logical reasoning. Includes file management, scheduling of requested processes, and handling peripheral devices.

Pause To implement a pause on the screen while programming. To freeze the screen, press CTRL/1. Repeat this procedure to continue. To freeze the screen briefly while the operator using your program reads a message, use a delay loop.

Pixel Contraction of the words "picture cell." The smallest addressable unit in a video graphics display. Pixels are very tiny and close to each other; thousands of them, viewed together, look like a solid picture when you are at a distance from the TV or monitor.

Player/Missile Graphics Describes figures that are designed by the programmer; an advanced programming technique. These special figures are generally

used for animation. The players and/or missiles do not affect the background (setting, etc.) when they move on the screen. The information for them is stored in a separate section of computer memory that is set aside by the programmer. P/M graphics is Atari's method of creating an image that appears two-dimensional on the screen, but is one-dimensional in RAM.

PLOT Atari BASIC command. Displays a given point in the graphics window, using data as defined by a previous **COLOR** command.

Plotting Mode A graphics mode in which you direct the computer to place a dot of a certain color at a point on the grid; in this way, you can draw pictures.

POKE Atari BASIC command that will address a certain location in the computer which issues instructions. A **POKE** places a number in a given RAM memory location; if a number is already at that location, it will be replaced by the new one.

POSITION (POS.) Atari BASIC command. Locates the cursor at a given position on the screen. It can be used in all graphics modes and is effective for the next input/output command that affects the graphics screen.

Positioning a Message In programming, positioning is when you tell the computer the column and row coordinates of your message. See **POSITION**.

PRINT Atari BASIC command. **PRINT** causes arithmetic expressions, string expressions, and data between quotation marks (" ") to be printed serially as listed.

Program A sequence of instructions—from the user to the computer—that specifies a process for manipulating data. Programs can be written in many languages of varying levels.

Prompt A visual signal from the program to "prompt" the operator to do something. The prompt should give some clue as to what the program requires (or allows) the user to do in response to it. The "READY" prompt from BASIC indicates that you can now enter a BASIC command or statement.

RAM Random Access Memory. This is the general purpose, erasable, and reusable memory located inside your Atari. The word memory, used by itself, is usually a reference to RAM.

REMark (REM) Statement **REMark** statements in programming begin with "REM" and are always ignored by the computer. A **REM** statement is used to label or title various sections of the program. **REMs** are there only for the user to see and to help the programmer remember what the different parts of the program are for.

ROM Read-Only Memory. **ROM** is memory that can be written only once and contains fixed data, such as the Atari 10K Operating System and the Atari BASIC language cartridge. **ROM** can be used only as input to the Atari computer. As opposed to **RAM**, **ROM** cannot be erased and reused.

Row The invisible line of boxes stacked horizontally in a grid. The Y-coordinate of a grid.

RUN Atari BASIC command. Followed by the file specification of a tokenized program, this command tells the computer to execute the program. RUN alone will cause the execution of the RAM-resident program.

SAVE Atari BASIC command. SAVE, followed by the file specification, outputs the program in the computer's memory in tokenized form to a disk. To RUN the program later, you must LOAD it from the disk or cassette on which it's SAVED, back into the Atari's memory. The SAVE instruction doesn't alter your program in memory. Always keep in mind that when you write a BASIC program, it will be lost (erased) unless you SAVE it before turning off the Atari, going to DOS, or using the NEW command.

Screen The surface of a monitor or TV set on which symbols and characters are displayed.

Semaphore Key (☐) Special key on the Atari XL keyboard that, when pressed, causes the character set to be displayed on the monitor in inverse video.

String An ordered sequence of data items, such as characters. For example, the word "string" is a string of six characters. See Variable.

String Variable An ordered sequence of numbers, characters, symbols, or a combination thereof, that may (but not necessarily) change during its use within a program. See DIM and Variable.

SYSTEM RESET Key Pressing the SYSTEM RESET console button will reset various key system locations. Depending on the situation, this key will cause a program to halt, restart, or cause what is known as a cold start. Pressing the SYSTEM RESET key for the purpose of returning to the beginning of a program, system, or application, results in what is called rebooting (resetting) with a warm start.

Text Mode GR.0, GR.1, and GR.2 are Atari text modes; these are the screens on which you can write.

Text Window (TW) Graphics modes 1 through 8 (and 14 and 15 on XL machines) have a text window for writing messages (text) on the lower part of the screen. The TW is actually composed of four lines of GR. 0 at the bottom of the graphics mode screen you're using. Since drawing modes won't allow you to write on the drawing screen, the text window lets you print whatever message you want to convey to the user in those four lines.

THEN Atari BASIC statement. THEN is the closing portion of the IF...THEN statement. See IF.

Variable A symbolically named entity which may assume an assigned value or a number of values. A variable is any quantity, numeric or string, which may (but not necessarily) change during the course of its use within a program. There are

limitations to be considered when naming your variables. For example, a dollar sign (\$) must be appended directly at the end of the name as a type declaration character for string variables; numeric variables need no type declaration character. The only other difference in the mechanics of using string variables as opposed to numeric variables is that string variables must be DIMensioned.

? The abbreviation for PRINT in BASIC.

* In BASIC, a special character indicating the times (multiplication) sign.

****Definitions excerpted from *The Atari User's Encyclopedia*, by Gary Phillips and Jerry White, copyrighted by The Book Company, 1984.**

Books for

1	2	3
		
4	5	6
		
7	8	9
		

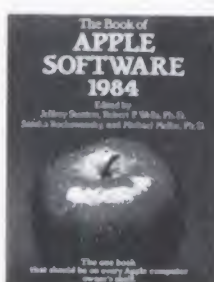
Books For

1



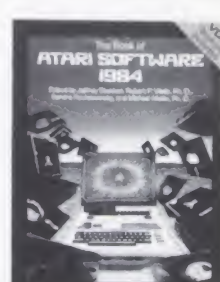
The Book of IBM Software 1984
 Edited by Robert P. Wells, Ph.D., Sandra Rochowansky and Michael Mellin, Ph.D.
 ISBN 0-912003-02-2
 \$19.95

2



The Book of Apple Software 1984
 Edited by Jeffrey Stanton, Robert P. Wells, Ph.D., Sandra Rochowansky and Michael Mellin, Ph.D.
 ISBN 0-912003-03-0
 \$19.95

3



The Book of Atari Software 1984
 Edited by Jeffrey Stanton, Robert P. Wells, Ph.D., Sandra Rochowansky, and Michael Mellin, Ph.D.
 ISBN 0-912003-04-9
 \$19.95

7



WHAT IF...? A Guide to Computer Modeling
 By Tom Simondi
 ISBN 0-912003-00-6
 \$19.95

8



Apple Graphics and Arcade Game Design
 By Jeffrey Stanton
 ISBN 0-912003-01-4
 \$19.95

9



Atari Graphics and Arcade Game Design
 By Jeffrey Stanton with Dan Pinal
 ISBN 0-912003-05-7
 \$16.95

13



The Texas Instruments User's Encyclopedia
 (TI-99, 2, 4, 4A and 8)
 By Gary Phillips and David Reese
 ISBN 0-912003-15-4
 \$14.95

14



The Apple User's Encyclopedia
 (Apple II, II+, IIe, III)
 By Gary Phillips, Joyce Conklin and Donald J. Scellato
 ISBN 0-912003-14-6
 \$19.95

15



The Atari User's Encyclopedia
 By Gary Phillips and Jerry White
 ISBN 0-912003-17-0
 \$19.95

All Reasons

4



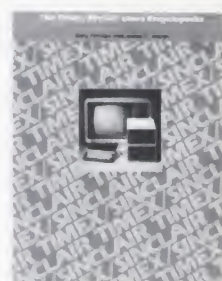
**Commodore 64
Color Graphics: A
Beginner's Guide**
By Shaffer &
Shaffer Applied
Research &
Development
ISBN 0-912003-06-5
\$14.95

5



**The TRS-80 User's
Encyclopedia
(Models I, III
and 4)**
By Gary Phillips
and James E.
Potter
ISBN 0-912003-12-X
\$19.95

6



**The Timex/
Sinclair User's
Encyclopedia
(TS1000, TS2068
and ZX81)**
By Gary Phillips
and James C.
March
ISBN 0-912003-16-2
\$14.95

10



**Commodore 64
Color Graphics:
An Advanced
Guide**
By Shaffer &
Shaffer Applied
Research &
Development
ISBN 0-912003-07-3
\$14.95

11



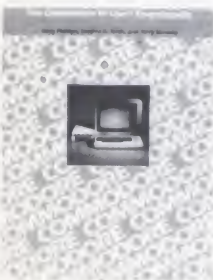
**The TRS-80 User's
Encyclopedia
(Model 100)**
By Gary Phillips,
Jacquelyn Smith
and Julia
Menapace
ISBN 0-912003-13-8
\$14.95

12



**The TRS-80 User's
Encyclopedia
(Color Computer
and MC-10)**
By Gary Phillips
and Guier S.
Wright III
ISBN 0-912003-11-1
\$14.95

16



**The Commodore
64 User's
Encyclopedia**
By Gary Phillips,
Sanjiva K. Nath
and Terry Silveria
ISBN 0-912003-10-3
\$14.95

17



**The Commodore
VIC-20 User's
Encyclopedia**
By Gary Phillips,
Roger Tierce,
Sanjiva K. Nath
and Terry Silveria
ISBN 0-912003-09-X
\$14.95

ARRAYS, INC.

ARRAYS, INC./THE BOOK DIVISION
11223 So. Hindry Ave. • Los Angeles, CA 90045



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 6 LOS ANGELES, CA

POSTAGE WILL BE PAID BY ADDRESSEE

ARRAYS, INC./THE BOOK DIVISION
Dept. G-1
11223 South Hindry Avenue
Los Angeles, CA 90045

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



ORDER FORM

FOLD HERE

Available at computer stores everywhere or directly from Arrays, Inc.

Title	Qty.	Price	Total
1. The Book of IBM Software 1984		\$19.95	
2. The Book of Apple Software 1984		\$19.95	
3. The Book of Atari Software 1984		\$19.95	
4. Commodore 64 Color Graphics: A Beginner's Guide		\$14.95	
5. The TRS-80 User's Encyclopedia (Models I, III, 4)		\$19.95	
6. The Times/Sinclair User's Encyclopedia (TS1000, TS2068, ZX81)		\$14.95	
7. WHAT IF...? A Guide to Computer Modeling		\$19.95	
8. Apple Graphics and Arcade Game Design		\$19.95	
9. Atari Graphics and Arcade Game Design		\$16.95	
10. Commodore 64 Color Graphics: An Advanced Guide		\$14.95	
11. The TRS-80 User's Encyclopedia (Model 100)		\$14.95	
12. The TRS-80 User's Encyclopedia (Color Computer and MC-10)		\$14.95	
13. The Texas Instruments User's Encyclopedia (TI-99/2.4.4A)		\$14.95	
14. The Apple User's Encyclopedia (Apple II, II+, IIe, III)		\$19.95	
15. The Atari User's Encyclopedia		\$19.95	
16. The Commodore 64 User's Encyclopedia		\$14.95	
17. The Commodore VIC-20 User's Encyclopedia		\$14.95	
18. Tips on Buying Software - N/C w/ purchase of any of the above books.		N/C	

FOLD HERE

Available at computer stores everywhere.

Or, order directly from Arrays, Inc./The Book Division using this postage paid envelope/order form.

Visa, MasterCard, Check or Money Order accepted

Name

Address

City

State

Zip

☐ Visa ☐ MasterCard

Card #

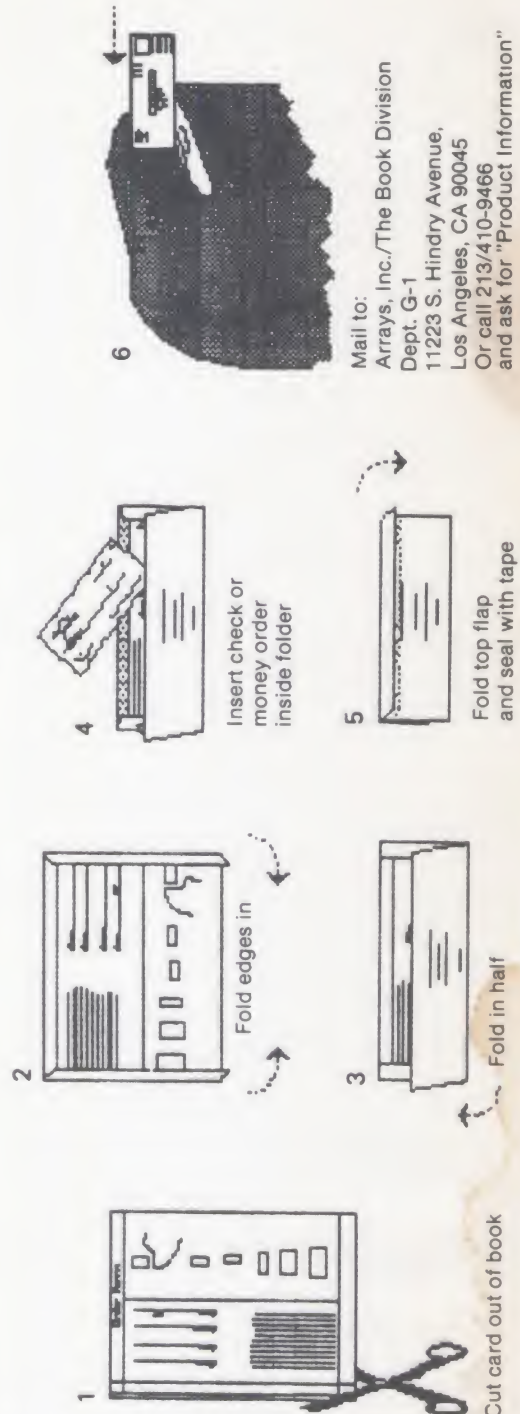
Exp. Date

Signature

Total Amount Enclosed \$_____ (California residents add 6% sales tax. Postage will be paid by Arrays, Inc./The Book Division).

Make checks payable to: Arrays, Inc.

FOLD HERE



ATARI COLOR GRAPHICS: A BEGINNER'S WORKBOOK

\$12.95
U.S.A.

Discover the exciting world of electronic art! The **Beginner's Workbook** is your invaluable guide to programming color graphics on your Atari home computer. It is a clear, step-by-step introduction to programming and graphics on both newer and older model Ataris.

No other personal computer even approaches the Atari's amazing versatility and speed in producing sophisticated graphics. This handy volume teaches you—at your own pace—design, color, and BASIC programming techniques that will help you make the most of your machine's extraordinary capacities.

Even a complete novice can easily follow all programs because they are so thoroughly explained. After the exercises, helpful review sections reinforce programming concepts, and numerous photographs and illustrations aid understanding.

The author presents many time-saving tips for speeding up your graphics. Easy-to-understand instructions help you master high resolution graphics in a jiffy.

The **Beginner's Workbook** is brimming with attractive graphics programs that make experimenting with color on your Atari a fun-filled adventure. This collection of enjoyable and instructive programs will turn you into a computer Picasso before you know it!

Also of interest to Atari owners:

The Atari User's Encyclopedia	\$19.95
The Book of Atari Software	\$19.95
Atari Graphics and Arcade Game Design	\$16.95

Available at computer outlets and bookstores, or directly from
Arrays, Inc., The Book Division.

ARRAYS, INC.

The Book Division

ISBN 0-912003-19-7